SIGMOD Officers, Committees, and Awardees

SIGMOD Edgar F. Codd Innovations Award

For innovative and highly significant contributions of enduring value to the development, understanding, or use of database systems and databases. Recipients of the award are the following:

Michael Stonebraker (1992)	Jim Gray (1993)	Philip Bernstein (1994)
David DeWitt (1995)	C. Mohan (1996)	David Maier (1997)
Serge Abiteboul (1998)	Hector Garcia-Molina (1999)	Rakesh Agrawal (2000)
Rudolf Bayer (2001)	Patricia Selinger (2002)	Don Chamberlin (2003)
Ronald Fagin (2004)	Michael Carey (2005)	Jeffrey D. Ullman (2006)
Jennifer Widom (2007)	Moshe Y. Vardi (2008)	Masaru Kitsuregawa (2009)
Umeshwar Dayal (2010)	Surajit Chaudhuri (2011)	Bruce Lindsay (2012)
Stefano Ceri (2013)	Martin Kersten (2014)	Laura Haas (2015)
Gerhard Weikum (2016)	Goetz Graefe (2017)	Raghu Ramakrishnan (2018)

SIGMOD Systems Award

For technical contributions that have had significant impact on the theory or practice of large-scale data management systems.

Michael Stonebraker and Lawrence Rowe (2015)   Martin Kersten (2016)   Richard Hipp (2017)
Jeff Hammerbacher, Ashish Thusoo, Joydeep Sen Sarma; Christopher Olston, Benjamin Reed, Utkarsh Srivastava (2018)

SIGMOD Contributions Award

For significant contributions to the field of database systems through research funding, education, and professional services. Recipients of the award are the following:

Maria Zemankova (1992)	Gio Wiederhold (1995)	Yahiko Kambayashi (1995)
Jeffrey Ullman (1996)	Avi Silberschatz (1997)	Won Kim (1998)
Raghu Ramakrishnan (1999)	Michael Carey (2000)	Laura Haas (2000)
Daniel Rosenkrantz (2001)	Richard Snodgrass (2002)	Michael Ley (2003)
Surajit Chaudhuri (2004)	Hongjun Lu (2005)	Tamer Özsu (2006)
Hans-Jörg Schek (2007)	Klaus R. Dittrich (2008)	Beng Chin Ooi (2009)
David Lomet (2010)	Gerhard Weikum (2011)	Marianne Winslett (2012)
H.V. Jagadish (2013)	Kyu-Young Whang (2014)	Curtis Dyreson (2015)
Samuel Madden (2016)	Yannis E. Ioannidis (2017)	Z. Meral Özsoyoğlu (2018)

SIGMOD Jim Gray Doctoral Dissertation Award

SIGMOD has established the annual SIGMOD Jim Gray Doctoral Dissertation Award to *recognize excellent research by doctoral candidates in the database field.* Recipients of the award are the following:

- **2006** *Winner*: Gerome Miklau. *Honorable Mentions*: Marcelo Arenas and Yanlei Diao.
- **2007** *Winner*: Boon Thau Loo. *Honorable Mentions*: Xifeng Yan and Martin Theobald.
- **2008** *Winner*: Ariel Fuxman. *Honorable Mentions*: Cong Yu and Nilesh Dalvi.
- **2009** *Winner*: Daniel Abadi. *Honorable Mentions*: Bee-Chung Chen and Ashwin Machanavajjhala.
- **2010** *Winner:* Christopher Ré. *Honorable Mentions*: Soumyadeb Mitra and Fabian Suchanek.
- **2011** *Winner*: Stratos Idreos. *Honorable Mentions*: Todd Green and Karl Schnaitterz.
- **2012** *Winner*: Ryan Johnson. *Honorable Mention*: Bogdan Alexe.
- **2013** *Winner*: Sudipto Das, *Honorable Mention*: Herodotos Herodotou and Wenchao Zhou.
- **2014** *Winners*: Aditya Parameswaran and Andy Pavlo.
- **2015** *Winner*: Alexander Thomson. *Honorable Mentions*: Marina Drosou and Karthik Ramachandra
- **2016** *Winner*: Paris Koutris. *Honorable Mentions*: Pinar Tozun and Alvin Cheung
- **2017** *Winner*: Peter Bailis. *Honorable Mention*: Immanuel Trummer
- **2018** *Winner*: Viktor Leis. *Honorable Mention*: Luis Galárraga and Yongjoo Park

A complete list of all SIGMOD Awards is available at: **https://sigmod.org/sigmod-awards/**

[Last updated : June 30, 2018]

Guest Editor's Notes

Welcome to the March 2019 issue of the ACM SIGMOD Record!

The new year of 2019 begins with a special issue on the **2018 ACM SIGMOD Research Highlight Award**. This is an award for the database community to showcase a set of research projects that exemplify core database research. In particular, these projects address an important problem, represent a definitive milestone in solving the problem, and have the potential of significant impact. This award also aims to make the selected works widely known in the database community, to our industry partners, and to the broader ACM community.

The award committee and editorial board included Yanlei Diao, Zack Ives, Wim Martens, Jun Yang, and Divesh Srivastava. We solicited articles from PODS 2018, SIGMOD 2018, VLDB 2018, ICDE 2018, EDBT 2018, and ICDT 2018, as well as from community nominations. Through a careful review process nine articles were finally selected as 2018 Research Highlights. The authors of each article worked closely with an associate editor to rewrite the article into a compact 8-page format, and improved it to appeal to the broad data management community. In addition, each research highlight is accompanied by a one-page technical perspective written by our associate editor or an external expert on the topic presented in the article. The technical perspective provides the reader with an overview of the background, the motivation, and the key innovation of the featured research highlight, as well as its scientific and practical significance.

The 2018 research highlights cover a broad set of topics, including (a) a combination of applied and theoretical research to understand why regular path queries in graph database applications behave better than worst-case complexity results suggest ("Bridging Theory and Practice with Query Log Analysis"); (b) a novel programming framework and system for systemizing the implementation of privacy algorithms. ("ϵktelo: A Framework for Defining Differentially-Private Computations"); (c) a principled approach to learn and reason about the entity matching classification task over a vector of similarity scores ("Entity Matching with Quality and Error Guarantees"); (d) a rigorous demonstration that theoretical ideas for enumerating the answers to a query can actually work in practice and deal with updates to the data ("Efficient Query Processing for Dynamically Changing Datasets"); (e) an innovative use of database techniques for scalable processing of massive datasets to solve the general problem of signal reconstruction ("Efficient Signal Reconstruction for a Broad Range of Applications"); (f) modeling the interaction between humans and data systems to satisfy the user's information need as a cooperative two-player game, where the strategy to play this game is learned through reinforcement learning ("How Do Humans and Data Systems Establish a Common Query Language?"); (g) a first study of the expressive power of linear algebra, used in machine learning algorithms, and how it relates to that of the relational algebra ("MATLANG: Matrix operations and their expressive power"); (h) the first algorithms for random sampling from data streams in the time decay model ("Online Model Management via Temporally Biased Sampling"); and (i) a new succinct data structure that can filter for point queries, range queries, and approximate counts efficiently while balancing the various hardware and workload trade-offs ("Succinct Range Filters").
On behalf of the SIGMOD Record Editorial Board, I hope that you enjoy reading the March 2019 issue of the SIGMOD Record!

Divesh Srivastava

March 2019

Your submissions to the SIGMOD Record are welcome via the submission site:
http://sigmod.hosting.acm.org/record

Prior to submission, please read the Editorial Policy on the website of the SIGMOD Record:
http://sigmod.org/sigmodrecord/authors/

Past SIGMOD Record Editors:

Ioana Manolescu (2009-2013)	Alexandros Labrinidis (2007–2009)	Mario Nascimento (2005–2007)
Ling Liu (2000–2004)	Michael Franklin (1996–2000)	Jennifer Widom (1995–1996)
Arie Segev (1989–1995)	Margaret H. Dunham (1986–1988)	Jon D. Clark (1984–1985)
Thomas J. Cook (1981–1983)	Douglas S. Kerr (1976-1978)	Randall Rustin (1974-1975)
Daniel O'Connell (1971–1973)	Harrison R. Morse (1969)	

Research Highlights: Bridging Theory and Practice with Query Log Analysis

Leonid Libkin
School of Informatics, University of Edinburgh
libkin@inf.ed.ac.uk

Take a database conference paper and search for *"in the real world"* in it; chances are high you will find it. Of course what is real depends on one's perspective: for a pure theory paper it could be what one saw in a systems paper, for a systems paper it could be an issue that implementors of DBMSs had to deal with, and for the latter it may be what the customers need. But to sharpen our research tools, it helps tremendously to understand that the real "real world" is, and adjust our (sometimes very elaborate) techniques to address problems that actually occur.

A nice example of this is analyzing the computational complexity of database queries. Database theory has developed an arsenal of tools for this. We know that for many classes of queries, the complexity is roughly $\|D\|^{O(\|Q\|)}$ for a database D and a query Q, where $\| \|$ means size. Thus, much research went into the detailed analysis of the structure of queries that removes $\|Q\|$ from the exponent and replaces it by a small fixed constant. We have a very good theoretical understanding of such classes, but we know much less about their relationship with queries that real-world users write.

Sometimes the situation is even more dramatic and the complexity of best known algorithms is exponential, e.g., $c^{\|D\|}$ for a constant c. This looks like an non-starter, and theoretical research tends to dismiss queries of such a complexity. But it shouldn't. To start with, we have many examples of problems working well in the real "real world" and yet having bad theoretical behavior. For example, satisfiability is the canonical NP-complete problem, and yet SAT solvers do very well these days. There are programming languages whose type inference algorithms require exponential time. And closer to databases, the current leader among graph databases, Neo4j, uses an NP-hard query evaluation algorithm and yet it works well, as is evidenced by their position in the graph database market.

The reason all these work well in practice is that the complexity analysis considers the worst case, and the worst is not necessarily what occurs in life. Yes, one can choke a SAT solver, one can write a simple program whose type will fill pages, and one can write a graph database query that will take forever even on a relatively small graph, but these are not the typical cases. The question then arises: what are the typical cases that one deals with in that real world?

This is the question that the paper by Wim Martens and Tina Trautner answers for path queries over graphs. To understand what the real world looks like, one has to observe it, rather than just make assumptions about it. This is what they do, by analyzing very large repositories of available SPARQL queries, and seeing what types of path queries they use. It is important to look at large query logs, and also at different ones, as different logs may well have very different characteristics and one shouldn't be making assumptions about the entire world out there based on a partial view of it, even if this view has many data points (this point is probably relevant even beyond analyzing query logs...).

The paper does not stop at analyzing query logs. It takes the analysis further to answer the following question: why some graph database queries, while behaving so badly in theory, actually do well in practice? The type of restrictions in path queries on graphs one deals with most often is in the mode in which paths are traversed: either there are no repeated nodes in paths (i.e., we have simple paths), or there are no repeated edges (such paths are called trails; this is the approach that Neo4j takes). Both have long been known to be NP-hard, in the worst case. From practice, we know that the trail semantics of Neo4j works well. Why?

The paper answers this question. By analyzing queries found in the logs, it establishes conditions that simultaneously cover a vast majority of real life queries, and at the same time admit efficient evaluation algorithms. Thus, the worst case might occur, but it is not that common, and this tells us that the high theoretical complexity is not seen in the real world.

This is a very nice *combination* of applied and theoretical research. One does not stop at simply analyzing the logs; instead the authors turn the result of the analysis into a nontrivial theoretical result that says when query evaluation is efficient.

I very much hope we shall see more papers of this kind. Very often the gap between theory and systems is too large in our community: theoreticians produce results motivated by their theoretical value, and systems research often finds such results too far fetched and goes ahead disregarding theoretical developments, hoping for the best essentially (Neo4j's NP-hard algorithm is an example: their approach would have been dismissed by theoreticians, but they went ahead, and it worked). To close the gap, we need to establish this back-and-forth between theory and systems, and the paper you are about to read is one of relatively few but very prominent examples of it. I hope more will follow.

Bridging Theory and Practice with Query Log Analysis

Wim Martens
University of Bayreuth
wim.martens@uni-bayreuth.de

Tina Trautner
University of Bayreuth
tina.trautner@uni-bayreuth.de

ABSTRACT

Since large structured query logs have recently become available, we have a new opportunity to gain insights in the types of queries that users ask. Even though such logs can be quite volatile, there are various new observations that can be made about the structure of queries inside them, on which we report here. Furthermore, building on an extensive analysis that has been done on such logs, we were able to provide a theoretical explanation why *regular path queries* in graph database applications behave better than worst-case complexity results suggest at first sight.

1. INTRODUCTION

The recent availability of large logs of structured queries provides new research opportunities for the database community. With millions of queries available for analysis, we suddenly have a large amount of information that can help us to identify interesting characteristics of real-world database queries. Such characteristics can then guide our focus when we want to study certain aspects of query evaluation or optimization, or if we simply want to understand the types of questions that users find interesting.

Database research has traditionally always had a strong focus on searching for subclasses of query languages that exhibit favorable computational properties. Well-known examples are the focus on *conjunctive queries* instead of full-fledged query languages, Datalog as a subset of Prolog, myriads of fragments of XPath or XQuery in the times of XML research, or even set semantics of queries (i.e., *select distinct* queries), as opposed to bag semantics.

Now that query logs are becoming available, we are obtaining hard data against which we can test or justify the importance of some of the specific problems we have been studying, and in which we may be able to discover new interesting cases. A nice side-effect for researchers is that, once we find a specific property of queries to be very prominent in logs, we immediately have numbers that we can use to motivate research on this specific property.

This paper is primarily based on the paper "Evaluation and Enumeration Problems for Regular Path Queries" (published in ICDT 2018), but also on "An Analytical Study of Large SPARQL Query Logs" (published in VLDB 2018)

However, there is also a snag: the query logs we currently have exhibit very different characteristics depending on the data source that is queried [5]. Due to the mixture of robotic and user-generated queries, these characteristics can even vary significantly on a day-to-day basis for the same query log [4]. This has some very important consequences that we need to keep in mind. First of all, we need to be careful that we don't focus too much on an aspect that is too specific, i.e., we would overfit our research. Needless to say, this is a delicate balance that is challenging to maintain. Second, studies on query log analysis should not be used to argue that some property of queries is not interesting. This is for the simple reason that, even though we may have "large" amounts of queries available, there is an even larger amount of queries that we do not have available (or will become important in the future) and we know nothing about. The old-fashioned elegance of a problem therefore remains extremely important for guiding research.

In this paper we report on some lessons learned from analysing over half a billion queries, coming from DBpedia, Semantic Web Dog Food, LinkedGeoData, BioPortal, OpenBioMed, British Museum, and WikiData. We also illustrate how some of the knowledge gained from this analysis could be used in theoretical research to give an explanation why certain types of queries seem to behave mostly unproblematically in practice even though their worst-case complexity is quite high.

2. ANALYSIS OF SPARQL LOGS

To the best of our knowledge, the first study on huge logs of structured queries was done by Bonifati et al. [5]. The study had a total of about 180M SPARQL queries, summarized in Table 1. The table mentions, for each of the logs, its total number of queries (*Total*) and the number of queries that could be parsed using Apache Jena 3.0.1 (*Valid*). From the latter set, duplicates were removed, resulting in the unique queries that could be parsed (*Unique*). The queries come from DBpedia, LinkedGeoData (LGD), BioPortal (BioP), OpenBioMed (BioMed), Semantic Web Dog Food (SWDF), British Museum (BritM), and WikiData. The `WikiData17` set is very small: it consists of the user-submitted example queries from Wikidata in February 2017. This first study has since been extended with 170M DBpedia queries [6] and 208M Wikidata queries [7], adding up to more than 550 million queries. In this paper we will

`https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples`

Table 1: Query logs in the corpus of [5].

Source	*Total #Q*	*Valid #Q*	*Unique #Q*
DBpedia9/12	28,534,301	27,097,467	13,437,966
DBpedia13	5,243,853	4,819,837	2,628,005
DBpedia14	37,219,788	33,996,480	17,217,448
DBpedia15	43,478,986	42,709,778	13,253,845
DBpedia16	15,098,176	14,687,869	4,369,781
LGD13	1,841,880	1,513,868	357,842
LGD14	1,999,961	1,929,130	628,640
BioP13	4,627,271	4,624,430	687,773
BioP14	26,438,933	26,404,710	2,191,152
BioMed13	883,374	882,809	27,030
SWDF13	13,762,797	13,618,017	1,229,759
BritM14	1,523,827	1,513,534	135,112
WikiData17	309	308	308
Total	180,653,910	173,798,237	56,164,661

primarily focus on the query logs in Table 1, but we will report insights from the other studies [7, 6] whenever relevant.

Size of Queries.

A first important discovery that was made in the logs is about the size of queries. In [5], this was measured by counting the number of *subject-predicate-object* triples in the queries, which are the SPARQL counterpart of atoms (or relational predicates) in relational databases. The distribution in these logs is extremely skewed: if we look in the *Unique* queries, over 56% have only a single triple. Even though there are queries with up to 229 triples, it is the case that up to six triples are enough to capture over 90% of the queries and, with up to twelve triples, we capture over 99%. In Wikidata logs that were recently investigated [7], the distribution is less skewed, at least for the *non-robotic* queries. Here, only 13% of the unique queries have a single triple. Moreover, one needs up to 9 triples to capture over 90% and up to 16 triples to capture over 99% of the queries.

Cyclicity.

These observations on the size of queries are important if we want to understand cyclicity. *Cyclicity* and *acyclicity* of queries is indeed a very important aspect of queries that has received a huge amount of attention in the literature (e.g., [11] and the references therein). An important reason is that queries in practice are assumed to be only *mildly cyclic*, which would be good news, since cyclic queries are more complex to evaluate. Since our standard definitions of cyclicity require a query to have at least three atoms to be cyclic [1], and since 78% of the unique queries in Table 1 only have up to two triples, we already know that the majority must be acyclic. But the assumption that real world queries are only mildly cyclic is also strongly confirmed when we look deeper. Out of all the conjunctive queries, even 99.9% are acyclic. Again, these numbers slightly shift when we look into the non-robotic Wikidata queries mentioned before, where around 97.8% of the unique queries investigated in [7] are acyclic. In terms of *treewidth*, we found queries in the logs with treewidth up to five (if one also allows property

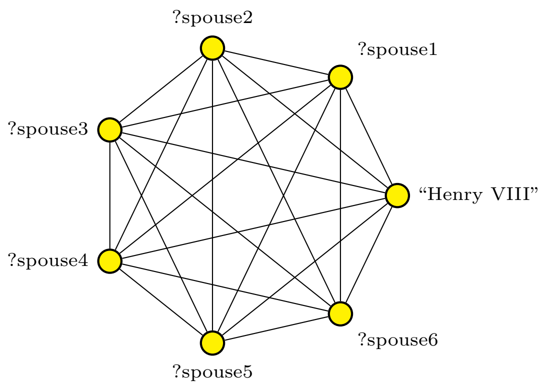


Figure 1: Graphical representation of a highly cyclic query: a 7-clique containing one constant and six variables. All edges connecting to Henry VIII are labeled "married-to" and all edges between the variables ?spouse1, . . . , ?spouse6 are labeled "! =". The query therefore searches for six different spouses of Henry VIIIth. The query is inspired on a query we found in the logs of Table 1.

paths). An interesting real-world query with treewidth five can be found in Figure 1. (The subquery consisting of the variables is a six-clique, which has treewidth five.)

Query Shapes.

Since so many queries in the logs are acyclic, it also makes sense to look more closely at their structure. More precisely, one can consider the graph structure induced by the subject-predicate-object triples in the queries by considering each triple (x, y, z) and turning it into two nodes x and z, connected by an undirected edge. (The construction of the graph is actually more subtle – sometimes *hypergraphs* are required. We refer to [5] for more details.)

For those queries that can be adequately represented as a graph, the *undirected* version of this graph was considered and it was investigated which fractions of the queries are a *single edge*, a *chain* (a connected, acyclic sequence of edges), a *star* (is a "central" node, to which chains can be attached), or a *tree*. These shapes are intended to be *cumulative*, so each shape generalizes the previous one. Considering the unique conjunctive queries, it turns out that around 78.98% are a single edge, 98.87% are a chain, 99.81% are a star, and 99.90% are trees. A visual inspection of the remaining queries showed that many of them can be seen as *flowers*, which are a central node, to which trees or *petals* can be attached. Here, a petal consists of two nodes u and v that are connected by chains. This generalization allowed to capture 99.94% of the queries. Most of the remaining queries consisted of multiple connected components. Generalizing from *flower* queries to *bouquet* queries allowed to capture essentially 100%. Here, a bouquet is a graph in which each connected component is a flower.

Differences Between Logs.

Even though some trends can be identified in the logs, there are also some drastic differences. This is a healthy warning for us: we should not declare that we now understand what users are interested in. For instance, in the

BioP13 and BioP14 logs, 79.66% and 40.48% of the unique queries use the GRAPH-operator, whereas this operator only occurs in 2.71% of the total queries. Bielefeldt et al. [4] observed huge differences in query volumes in Wikidata logs over different days, mainly due to automatically generated queries that, consequently, can have huge effect on the types of queries in the logs. Finally, in the data of Table 1, less than 1% of the queries use property paths, whereas this grows to 38% of the unique queries in [7].

3. CASE IN POINT: PATH QUERIES

Regular path queries (RPQs) are a crucial feature of graph database query languages, since they allow us to answer queries that involve arbitrarily long paths in graphs using regular expressions. We give an example. Consider the toy graph database in Figure 2, which is loosely inspired on a part of the Wikidata graph. Suppose that we want to find *artists who died at age 27*, we can easily do so using a regular path query. (These artists are known under the name "27 club". The club has famous members such as Kurt Cobain, Jimi Hendrix, Janis Joplin, Jim Morrison, and Amy Winehouse.) For instance, we can retrieve the persons who died at age 27 with a Cypher-like subquery of the form

```
CONSTRUCT (x)
  MATCH   (x:Person)-[:age-at-death]->(y:Integer)
    WHERE  y = 27
```

Likewise, artists can be found by the query

```
CONSTRUCT (x)
  MATCH   (x:Person)-[:occupation]->()
                     -[:subclassof*]->(y:Profession)
    WHERE  y.name = 'artist'
```

The second query asks for persons whose occupation is a profession that is connected with a subclassof-path to "artist". Here, we used the regular expression subclassof* to allow arbitrarily long paths in which every edge is labeled with subclassof. Since we may not know in advance how many subclassof-edges we have to consider, it is very comfortable to be able to use the regular path query subclassof*. The example also illustrates the robustness of regular path queries. Even when the graph database changes (e.g., by introducing an additional profession such as "string instrumentalist"), the query still returns the correct results.

Regular path queries or RPQs started as an academic idea in Cruz et al.'s seminal paper [8] and are nowadays part of SPARQL, Cypher and Oracle's PGQL. Although the main idea behind RPQs is always to match regular expressions against paths in a graph database, academic research and real-world systems do not always agree on how this should be done. The main difference lies in which paths should be considered for matching, and the most considered candidates are *all paths* or *paths without repeated nodes or edges*. Whereas academic research most commonly allows all paths (which allow polynomial time algorithms to test if a matching path exists between two given nodes), graph database systems usually revert to paths without repeated nodes or edges. There seem to be different reasons why this is so. First of all, this restriction always ensures that the number of paths that can match is finite, so one does not have to deal with infinity. Second, paths without repeated nodes or edges gives the semantics that some users seem to prefer [Lindaaker, personal communication]. From a theoretical point of view, however, such paths very quickly lead to intractability. Even testing if a matching path exists between nodes is NP-complete, see Theorem 1.

3.1 Complexity of Simple Paths and Trails

We briefly want to explain some of the fundamental results about RPQ evaluation against paths without repeated nodes or edges. We use edge-labeled graphs as abstractions for graph databases. To this end, let Σ be a set of *labels*. A graph database (with labels in Σ) is a pair $G = (V, E)$, where V is the finite set of *nodes* of G and $E \subseteq V \times \Sigma \times V$ is the set of *edges*. We say that edge $e = (u, a, v)$ is *from node u to node v* and *has label a*. Notice that this definition allows graphs to have self-loops and multiple edges from u to v if they have different labels. The *size* of a graph G, denoted by $|G|$, is defined as $|G| = |V| + |E|$.

A *path* from node u to node v in G is a sequence

$$p = (v_0, a_1, v_1)(v_1, a_2, v_2) \cdots (v_{n-1}, a_n, v_n)$$

of edges in G such that $u = v_0$ and $v = v_n$. By $\text{lab}(p)$ we denote the sequence $a_1 \cdots a_n$ of labels on the edges of p. The length of a path p is its number of edges. A path p is *simple* if it has no repeated nodes, that is, all nodes $v_0, \ldots, v_n$ are pairwise different. It is a *trail* if it has no repeated edges, that is, every edge appears only once in p.

Regular path queries are abstracted as regular expressions. Here, ε, and every Σ-symbol is a regular expression; and if r and s are regular expressions, then so are $(r \cdot s)$, $(r + s)$, and (r^*). (To improve readability, we use associativity and the standard priority rules to omit braces in regular expressions. We usually also omit the outermost braces.) We use $r?$ to abbreviate $r + \varepsilon$. The *size* $|r|$ of a regular expression is the number of occurrences of Σ-symbols in r. For example, $|((a \cdot b) \cdot a)^*| = 3$. We define the *language* $L(r)$ of r as usual. A path p *matches* r if $\text{lab}(p) \in L(r)$, that is, the sequence of labels on the edges of p is in the language of r. The following two decision problems are central to evaluation of regular path queries over simple paths and trails.

SimPath($\mathcal{R}$)	
Given:	A graph $G = (V, E)$, two nodes $x, y \in V$, and an RPQ $r \in \mathcal{R}$.
Question:	Is there a simple path from x to y in G that matches r?

Trail($\mathcal{R}$)	
Given:	A graph $G = (V, E)$, two nodes $x, y \in V$, and an RPQ $r \in \mathcal{R}$.
Question:	Is there a trail from x to y in G that matches r?

We parameterized the problems with a class $\mathcal{R}$ of regular expressions, so that we can discuss variants of these problems. (If $\mathcal{R}$ is just a single regular expression r, then we simply write SimPath(r) instead of SimPath($\{r\}$), and analogously for Trail.)

Notice that any algorithm that is able to answer RPQs (i.e., compute all matching paths) while considering simple paths and trails, is able to solve these decision problems. So, the complexity of these decision problems is important. Notice that both problems are trivially in NP. Mendelzon

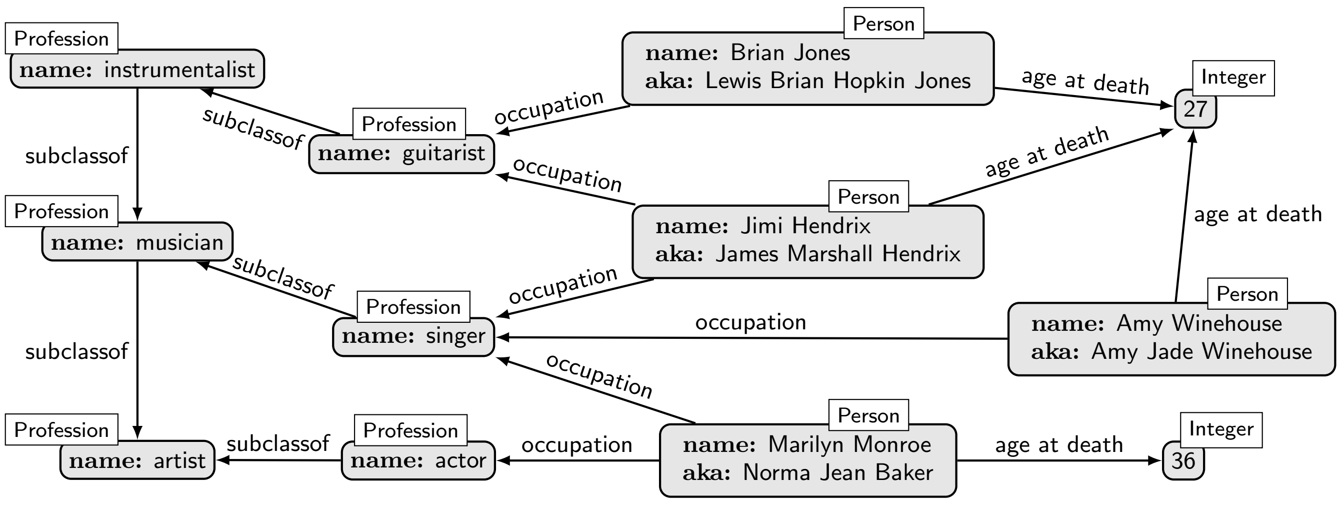


Figure 2: A graph database (as a *property graph*), inspired on a fragment of WikiData

and Wood studied SimPath and discovered that it becomes NP-hard very quickly [16]. Items (a) and (b) of the following theorem come from their work:

THEOREM 1. *The following problems are NP-complete:*
(a) SimPath$((aa)^*)$ *(b)* SimPath(a^*ba^*)
(c) Trail$((aa)^*)$ *(d)* Trail(a^*ba^*)

Items (c) and (d) can be obtained by easy reductions from the *two disjoint paths* problem, using the standard split-graph construction from Perl and Shiloach [18] or LaPaugh and Rivest [12] and the same reductions as for simple paths used by Mendelzon and Wood [16].

So, not only are SimPath and Trail NP-complete, they are even NP-complete in cases where the regular path query is fixed. Furthermore, the expressions for which this NP-completeness holds can be very small. It is therefore no surprise that, from a worst-case complexity perspective, it seems to be a bad idea to build a query language for graph databases on simple path or trail semantics. We note that it is understood for which fixed regular expressions SimPath and Trail are NP-complete [3, 13].

3.2 What About Query Logs?

Once query logs became available, we have been able to analyze what kind of RPQs actually occur. The study of Bonifati et al. [5] had 247k SPARQL property paths in unique queries, which gave us a first impression. Syntactically, SPARQL property paths are extensions of RPQs. This is important, because it means that the types of regular expressions we will see are not syntactically constrained by the query language. On top of the ordinary operators for RPQs, SPARQL allows operators for wildcards and for following edges in the reverse direction. This would not be the case for Cypher, for example. (In Neo4j's Cypher 3.2 manual, only single labels or wildcards were allowed below Kleene stars [17]. Cypher 9 is becoming more liberal and allows disjunction below a Kleene star, see [10, Figure 3: Syntax of Cypher patterns]. In the near future, Cypher plans to support full regular path queries [10].)

In Table 2, we provide a summary of the types of property paths found in the data of [5]. That is, Table 2 is not the table appearing in [5], but we went over the raw data again and aggregated the types of expressions slightly differently. We use the following conventions: (1) lower case letters denote single symbols, (2) upper case letters denote sets of symbols, (3) we denote a wildcard test by $\sqcup$, (4) we do not distinguish between following an edge in the forward or backward direction, (5) each expression type also encompasses its symmetric form. For instance, when we write a^*b, we count the expressions of the form a^*b and ba^*. We always list the variant that occurred most often in the data. That is, a^*b occurred more often than ba^*. These conventions are the same as in our conference paper [14].

Under *Expression Type*, the table summarizes which types of expressions are in Bonifati et al.'s data set, sometimes parameterized by a number ℓ for which the next column describes the values that were found. *Relative* describes which percentage of the 247,404 expressions fall into this expression type. We discuss *STE?* in the next section.

In Table 2 we can immediately observe that the property paths found in the query logs of Bonifati et al. are not very complex and that the expressions mentioned in Theorem 1 only occur very rarely. In fact, the query $(ab)^*$ occured only once and we found out that this query was posed by a theoretician testing the robustness of the engine [Vrgoč, personal communication].

Another thing to keep in mind is how to interpret the classification in Table 2. After all, property paths do not occur often in the logs of Table 1: only about 0.4% of the queries have them. However, this seems to be an artifact of the underlying data. Most of the property paths appear in DBpedia queries, but DBpedia was designed when property paths were not yet part of SPARQL. In a more recent study on Wikidata query logs, containing 35 million unique queries, a drastically larger 38.94% of the queries use property paths [7]. Moreover, the structure of these property paths shows a picture similar to what we see in Table 2 [7].

4. SIMPLE TRANSITIVE EXPRESSIONS

We now define a class of RPQs called *simple transitive expressions (STEs)*, with the intent of capturing the vast majority of the expressions in Table 2, while avoiding the problems discussed in Section 3.1. Intuitively, simple tran-

Expression Type	ℓ	*Relative*	*STE?*
$(a_1 + \cdots + a_\ell)^*$	2–4	29.10%	yes
$\sqcup$		25.48%	yes$^{(*)}$
a^*		19.66%	yes
$a_1 \cdots a_\ell$	2–6	8.66%	yes
a^*b		7.73%	yes
$(a_1 + \cdots + a_\ell)$	1–6	6.61%	yes
$(a_1 + \cdots + a_\ell)^+$	1–2	1.54%	yes
$a_1?a_2?\cdots a_\ell?$	1–5	1.15%	yes
$a(b_1 + b_2)?$		0.01%	yes
$a_1a_2?\cdots a_\ell?$	2–3	0.01%	yes
$a^*b?$		$< 0.01\%$	yes
abc^*		$< 0.01\%$	yes
$A_1 \cdots A_\ell$	2–6	$< 0.01\%$	yes
$(a_1 + a_2)?$		$< 0.01\%$	yes
$\sqcup^*$		$< 0.01\%$	yes$^{(*)}$
$\sqcup b^*$		$< 0.01\%$	yes$^{(*)}$
$\sqcup?$		$< 0.01\%$	yes$^{(*)}$
$(ab^*) + c$		$< 0.01\%$	no
$a^* + b$		$< 0.01\%$	no
$a + b^+$		$< 0.01\%$	no
$a^+ + b^+$		$< 0.01\%$	no
$(ab)^*$		$< 0.01\%$	no

Table 2: Structure of the 247,404 SPARQL property paths that were also used in the query logs investigated by Bonifati et al. [5]. The structure is sometimes in terms of a variable $\ell \in \mathbb{N}$, for which the second column indicated the values that were found in the logs. *Relative* indicates which percentage of the 247,404 property paths have this structure.

sitive expressions aim at capturing very basic navigation in graphs: first do some *local navigation*, followed by an optional *transitive step*, and finally again some *local navigation*. The rationale is that, if we want to connect entities in a graph database, then this is a natural way to navigate. Let us again consider our running example of artists that died at the age of 27. When we want to find out if a Person is an artist, we first need to do some local navigation (following an `occupation`-edge) and then perform a transitive reflexive step (following an arbitrarily long path of `subclassof`-edges). More precisely, simple transitive expressions allow to:

1. first follow a path of length *exactly* k_1 or *at most* k_1 (for some $k_1 \in \mathbb{N}$),
2. then do a (reflexive) transitive closure step,
3. finally, follow a path of length *exactly* k_2 or *at most* k_2 (for some $k_2 \in \mathbb{N}$).

All three steps are subject to label tests. Furthermore, any step can be omitted, so a simple transitive expression can also express that paths must have length between k_1 and $k_1 + k_2$. In the following definition, we use sets $A = \{a_1, \ldots, a_\ell\} \subseteq \Sigma$ to abbreviate disjunctions $(a_1 + \cdots + a_\ell)$.

Definition 2. *An* atomic expression *is of the form $A \subseteq \Sigma$ with $A \neq \emptyset$. A* bounded expression *is a regular expression of the form $A_1 \cdots A_k$ or $A_1? \cdots A_k?$, where $k \geq 0$ and each A_i is an atomic expression. Finally, a* simple transitive expression (STE) *is a regular expression*

$$B_{pre} T^* B_{suff},$$

where B_{pre} and B_{suff} are bounded expressions and T is ε or an atomic expression.

A minor technicality is that we can take $T = \varepsilon$. This means that T^* will only match the empty word, and therefore the STE defines a finite language. In Table 2 the column *STE?* indicates whether the expression is an STE. Here, we write "yes$^{(*)}$" to indicate that the expression is an STE if a wildcard is treated the same as a set of labels A. (Our algorithms indeed can be generalized to incorporate wildcards.)

In total, we saw that only 20 property paths are not STEs or trivially equivalent to an STE (by taking $T = \varepsilon$ in the definition of STEs, for example). For instance, the expression type $a_1a_2?\cdots a_\ell?$ is equivalent to an STE where $B_{\text{pre}} = a_1$, $T = \varepsilon$, and $B_{\text{suff}} = a_2?\cdots a_\ell?$. In this sense, 99.992% of the property paths in Table 2 correspond to STEs.

In fact, *all* expressions in the table except for $(ab)^*$ are unions of STEs. Unions of STEs can actually be handled in the same way than STEs, by applying the STE evaluation algorithm to each part of the union.

4.1 Dichotomies for STEs

Our main technical results are two dichotomies for evaluating STEs under simple path and trail semantics. That is, we precisely characterize for which classes $\mathcal{R}$ of STEs the problems SimPath for $\mathcal{R}$ and Trail for $\mathcal{R}$ are easy and for which classes these problems are difficult. Here, "easy" and "difficult" refer to complexities in parameterized complexity, namely *fixed-parameter tractable* and W[1]-hard. Our results will imply that SimPath and Trail are "easy" for the types of expressions in Table 2 — except for $(ab)^*$. Furthermore, the parameters on which the complexity can exponentially depend are small.

Some Examples and Intuition.

We give a bit of intuition about our results. Throughout the example, we use the following notation. The input graph is always denoted as G, and it has n nodes and m edges. We always denote the start and end nodes in the input of the SimPath problem by x and y, respectively. We will abbreviate long concatenations with a power notation, that is, we use r^k to denote a sequence of k times the expression r. For instance a^4 denotes the expression $aaaa$. Let a^k denote the class $\{a^k \mid k \in \mathbb{N}\}$ of STEs. We define the classes $(a?)^k$, $a^k a^*$, $ba^k a^*$, and $a^k ba^*$ analogously.

We now discuss the complexities of SimPath for these classes. As a first example, we consider SimPath for $(a?)^k$. This problem is easy to solve: one can simply use an algorithm that tests reachability with a-labeled edges. The crux is that loops do not matter: if there is a path from x to y that matches $(a?)^k$ then there is also a simple such path, since removing loops does not change matching $(a?)^k$.

This technique does not work for our second example: SimPath for a^k. However, Alon et al.'s color coding technique [2] can solve this problem in time $2^{O(k)} m \log n$. Color coding therefore shows that SimPath for a^k is fixed-parameter tractable, where the parameter is the size k of the RPQ: it is an algorithm with complexity $f(k) \cdot p(|G| + k)$, where f is a computable function and p is a polynomial. The function f is even single exponential in this case. Notice that, if P

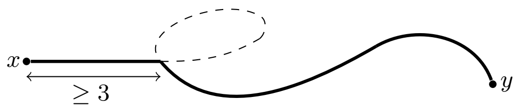


Figure 3: Intuition behind cuttability, using $bbba^*$

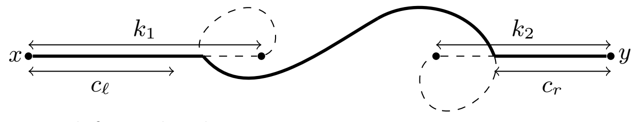
$\neq$ NP, we cannot hope for f to be a polynomial function, because SimPath for a^k is at least as difficult as the Hamiltonian Path problem. (Indeed, the cases of SimPath for a^k where we give a graph G with only a-labeled edges and the RPQ a^{m+1} are equivalent to the Hamiltonian Path problem for G.)

As a third example, we consider SimPath for a^ka^*. This problem requires yet another technique, since color coding is designed to work for fixed-length paths. It can be solved in time $2^{O(k)}(n^2 + mn)$, however, using the representative sets technique of Fomin et al. [9]. The representative sets technique is nontrivial and addresses the following problem. Assume that we try to deal with a^ka^* naively by considering all simple paths P of length k that start in x. For each such path P, assuming it ends in some node x_P, we could then test reachability from x_P to y while avoiding the nodes of P. But this algorithm is too inefficient. We may have up to n^k different possibilities for P, which means that the running time is not of the form $f(k) \cdot p(|G| + k)$ for a polynomial p and computable function f. In other words, it does not show that the problem is fixed-parameter tractable. This is where the representative sets technique is useful. It shows that the number of different paths P we have to consider can be limited to $2^{O(k)}n$, which makes the problem fixed-parameter tractable. The representative sets technique can even be adapted so that it *enumerates* all the simple paths.

We turn to two cases where the edge labels become important. First, consider SimPath(ba^ka^*). Here, we can simply enumerate all b-edges that start in x and then use the algorithm for SimPath(a^ka^*) from there (and making sure that we don't visit x). This shows that SimPath(ba^ka^*) is fixed-parameter tractable.

Second, take SimPath(a^kba^*). At its core, this problem is a variant of the Two Disjoint Paths problem. We are essentially searching for two nodes x' and y' such that there is a path P_1 of length k from x to x' and a path P_2 from y' to y. Moreover, P_1 and P_2 should be node-disjoint and there should be a b-edge from x' to y'. Since we can prove that this Two Disjoint Paths problem (with parameter k) is W[1]-hard [14], it turns out that SimPath(a^kba^*) is hard as well.

The central notion in our dichotomy for SimPath is *cut borders* of STEs. We explain this notion intuitively, based on two simple examples. Consider the expressions $r_1 = aaaa^*$ and $r_2 = aaab^*$. Assume that, as in Figure 3, we found a path p (that may contain a loop) from x to y that matches r_1. Intuitively, if we want to test if the simple path p' obtained from p by deleting all loops still matches r_1, we just need to test if p' has length at least three. For r_2, however, we additionally need to test that the loop does not occur in the prefix of length 3 of p. For this reason, the cut border of r_2 will be equal to 3. We can prove that this notion of cut border is indeed the crucial one for the complexity of SimPath.



c_ℓ : left cut border
c_r : right cut border

Figure 4: Assume $r = A_1 \cdots A_{k_1} T^* A'_{k_2} \cdots A'_1$ has left and right cut borders c_1 and c_2, respectively. Assume that an arbitrary path from s to t matches r such that its length k_1 prefix and length k_2 suffix do not have loops and are node disjoint. If, after removing all loops, (1) the length c_1 prefix and length c_2 suffix are still the same and (2) the path still has length at least $k_1 + k_2$, then it matches r.

Dichotomy for Simple Paths.

We now state our main result on SimPath and explain the cut borders, cuttability, and the sampling condition after its statement. (We only require the condition that $\mathcal{R}$ can be sampled for the lower bound proof in part (b).)

Theorem 3. *Let $\mathcal{R}$ be a class of STEs that can be sampled.*

(a) If $\mathcal{R}$ is cuttable, then SimPath($\mathcal{R}$) is solvable in time $2^{O(s)}n^{c+3}m$, where c is the cut border of $\mathcal{R}$.

(b) Otherwise, SimPath($\mathcal{R}$) cannot be solved in time $f(s) \cdot (n+m)^c$ for a constant c and a computable function f, unless FPT = W[1].

Here, n and m are the number of nodes and edges in the graph, respectively, and s is the size of the regular expression.

Here, FPT is the class of problems that is *fixed-parameter tractable*. It is a standard assumption in parameterized complexity theory that FPT $\neq$ W[1]. This assumption has a similar calibre as the P $\neq$ NP assumption in terms of decision problems.

We now explain cut borders, cuttability, and the condition that $\mathcal{R}$ can be sampled. To this end, the *left* (resp., *right*) *cut border* of an STE $r = A_1 \cdots A_{k_1} T^* A'_{k_2} \cdots A'_1$ is the largest value i such that T has a symbol that is not in A_i (resp., A'_i). If we have $A_1? \cdots A_{k_1}?$ (resp., $A'_{k_2}? \cdots A'_1?$), then the left (resp., right) cut border is 0. The *cut border* of r is the sum of its left and right cut border. A class $\mathcal{R}$ of STEs is *cuttable* if there exists a $c \in \mathbb{N}$ such that the cut border of each expression $r \in \mathcal{R}$ is at most c. The intuition of cut borders is explained in Figure 4: they characterize parts of paths in which it is not allowed to remove loops to obtain a simple path that still matches the expression.

Finally, we say that $\mathcal{R}$ *can be sampled* if there exists an algorithm that, given a number k in unary, returns an expression from $\mathcal{R}$ that has cut border at least k. Notice that this is a very weak restriction on $\mathcal{R}$.

Notice that the difference between cuttable and non-cuttable classes of STEs can be subtle. Using the same notation as with our previous examples, the classes a^kb^* and $a^k(a+b)^*$ are not cuttable, but $(a+b)^ka^*$ is. Looking back at Table 2, we see that abc^* is 2-bordered and all other STEs are either 0-bordered or 1-bordered. It therefore seems that cut borders in practice are small and over 99% of the expressions fall on the tractable side of Theorem 3.

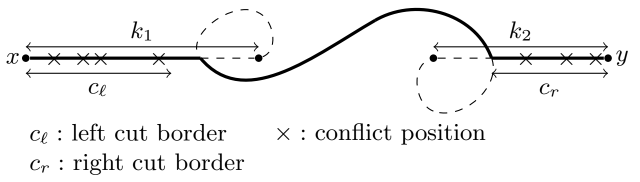


Figure 5: Visualization of the effect of conflict positions in a path that matches an STE r. If we would start with an arbitrary path and remove loops, we mainly need to be careful about labels behind the cut borders that can be identical to labels in the transitive part.

Dichotomy for Trails.

We now present a similar dichotomy for trails, obtained in [15]. The dichotomy is, perhaps surprisingly, different from the one in Theorem 3 in the sense that more classes fall on the tractable side. For instance, $\mathsf{SimPath}(a^kb^*)$ is intractable, whereas $\mathsf{Trail}(a^kb^*)$ is fixed parameter tractable because the a-path and the b-path can be evaluated independent of each other (no a-edge will be equal to a b-edge). We explain *conflict positions*, *almost conflict-freeness*, and *conflict-sampling* after the theorem statement. (The condition that $\mathcal{R}$ can be conflict-sampled is only needed for (b).)

THEOREM 4. *Let $\mathcal{R}$ be a class of STEs that can be conflict-sampled.*

(a) If $\mathcal{R}$ is almost conflict free, then $\mathsf{Trail}(\mathcal{R})$ is solvable in time $2^{O(s)} \cdot m^{c+6}$, where c is the number of conflict positions in $\mathcal{R}$.

(b) Otherwise, $\mathsf{Trail}(\mathcal{R})$ cannot be solved in time $f(s)(n+m)^c$ for a constant c and a computable function f, unless FPT = W[1].

Here, n and m are the number of nodes and edges in the graph, respectively, and s is the size of the regular expression.

If $r = A_1 \cdots A_{k_1} T^* A'_{k_2} \cdots A'_1$ is an STE, we say that a position left of the left cut border (resp., right of the right cut border) is a *conflict position* if T and A_i (resp., A'_i) have a common symbol or, equivalently, have a non-empty intersection. If we have $A_1? \cdots A_{k_1}?$ (resp., $A'_{k_2}? \cdots A'_1?$), then the left (resp., right) cut border is 0 and therefore there are no cut positions. In Figure 5 we give a visual intuition about the meaning of conflict positions. A class $\mathcal{R}$ of STEs is *almost conflict free* if there exists a constant $c \in \mathbb{N}$ such that each expression $r \in \mathcal{R}$ has at most c conflict positions. The class a^kb^* is not cuttable, but it is conflict-free because $\{a\}$ and $\{b\}$ have an empty intersection. The point is that an edge labeled by some symbol in $\{a\}$ can never be the same than an edge labeled by some symbol in $\{b\}$, since their labels must be different. Therefore, we can evaluate a^k and b^* separately.

A class $\mathcal{R}$ of STEs *can be conflict-sampled* if there exists an algorithm that, given a number k in unary, returns an expression $r \in \mathcal{R}$ with at least k conflict positions.

Extension to Enumeration of Paths.

Real-life graph databases are usually not primarily interested in solving decision problems, but in computing the answers to a query. *Enumeration algorithms* can be seen as a theoretical framework in which such problems can be studied. Such algorithms typically consider the *preprocessing time* and *delay* for computing the answers of a query. Here, the *preprocessing time* is the time required before producing the first answer (and possibly build a data structure so that consecutive answers can be generated quickly) and the *delay* is the time required between two consecutive answers. In this framework, the requirement is usually that each answer is returned only once.

The tractability results from Theorems 3(a) and 4(a) can be extended to enumeration problems. Using an adaptation of Yen's algorithm [19] that works with labeled simple paths (resp., labeled trails), it can be shown that the *paths that match the expressions* can also be enumerated in such a way that the *delay* between the answers roughly corresponds to the upper bounds in Theorems 3(a) and 4(a).

THEOREM 5. *Let $G = (V, E)$ be a graph, x and y be two nodes in V, and $\mathcal{R}$ a set of STEs.*

If $\mathcal{R}$ is cuttable, then the simple paths from x to y that match r, for a given $r \in \mathcal{R}$ can be enumerated with $2^{O(s)} \cdot n^{c+3}m$ preprocessing time and $2^{O(s)} \cdot n^{c+4}m$ delay.

If $\mathcal{R}$ is almost conflict free, then the trails from x to y that match r, for a given $r \in \mathcal{R}$ can be enumerated with $2^{O(s)} \cdot m^{c+6}$ preprocessing time and $2^{O(s)} \cdot m^{c+7}$ delay.

Core Techniques.

At the core of our tractability results lies the representative sets technique of Fomin et al. [9]. This technique can be used to find simple paths and trails of length at least k in time $2^{O(k)}(n^2 + nm)$, given a graph and the number k. If regular path queries are involved, the technique is only compatible with certain languages, such as cuttable or conflict-free STEs. The compatible languages have the property that we only need to guard a constant number of nodes/edges at the beginning and at the end of the path, to make sure that the rest of the path does not re-use the same nodes/edges.

Indeed, we can show that for languages violating this property, the problem becomes intractable. The reason is that it becomes at least as hard as a parameterized version of the two-disjoint paths problem. This parameterized problem asks: given a graph G, node pairs (x_1, y_1) and (x_2, y_2), and parameter $k \in \mathbb{N}$, are there two disjoint paths p_1 from x_1 to y_1 and p_2 from x_2 to y_2 such that p_1 has length k. (One can consider node-disjoint or edge-disjoint paths here.) We prove that this problem is W[1]-hard, both when node- or edge disjointness is required.

4.2 What Does This Mean?

If we interpret Theorems 3 and 4 in the light of the real world property paths in Table 2 we can observe the following. Let n and m be the number of nodes and edges of the graph, respectively.

Concerning simple paths, Theorem 3 gives us a running time of $2^{O(s)}n^{c+3}m$ for regular path query evaluation, where s is the size of the regular path query and c is the cut border. This result, together with the observation that the largest cut border in Table 2 is two, and therefore very small, can be seen as an explanation why, in practice, simple path semantics usually does not bring systems to their knees, even though this would theoretically be possible using regular expressions such as $(aa)^*$. Since the evaluation problem under simple path semantics generalizes the Hamilton Path problem (if $s = n - 1$), we cannot hope for a significantly better complexity unless P = NP.

One should keep in mind that this is a worst-case bound. In most practical settings, we expect that the run-time of even more naive evaluation algorithms will not come close to requiring n^{c+3} time for these simple expressions. For instance, the n^c factor comes from considering all paths that start in a given node x and obey a label constraint. For instance, for the expression abc^*, these are just the paths that start in x and are labeled ab. While this can, in the worst case, be n^2 many paths, we expect this to be much less in real databases.

The story for trails is similar. Here our upper bound admittedly gives less efficiency guarantees than the one for simple paths, but this is mainly because we have developed our methods for simple paths and then adapted them for trails. Furthermore, the dichotomy shows that it is easier to deal with trails than with simple paths: for every class of queries for which we have fixed-parameter tractable algorithms for simple path semantics, we also have them for trail semantics, but not vice versa.

5. CONCLUSIONS

The results in Section 4 can be seen as a theoretical explanation why evaluating certain queries (regular path queries against simple paths and trails) in graph databases seems to be less problematic in practice than theoretical results seem to suggest. The main reason is that, in the query logs that were considered in Section 2, the parameters that have a drastic impact on the complexity of evaluation remain relatively small.

In this sense, this paper showcases a line of work in which query log analysis was useful. However, a lot of work still remains to be done. First of all, the analysis of the logs in Section 2 showed much more than just the distribution of regular path queries. For instance, the shapes of queries found in the logs may be useful to generate realistic benchmarks. Second of all, the query log analysis from Section 2 itself is challenging too. For instance, in all the query logs we have seen until now, the distribution of the queries (and of interesting properties of queries) is extremely skewed. It is not clear how we balance finding interesting aspects of queries in logs with the fact that so many queries are extremely small, e.g., only have a single triple.

Furthermore, apart from having investigated successful and timeout queries for Wikidata [7], we do not know much about the combination of queries and data. For instance, it could be very interesting to study which parts of the graph are used for the evaluation of a query, and how large intermediate results become. Such studies must be left to future work.

Acknowledgments

In terms of methodology, we were heavily inspired by a line of work initiated by Frank Neven. In 2004, Frank had the idea to do a practical study on the shapes of regular expressions in schemas for XML data. This study motivated theoretical work on simple regular expressions (later called chain regular expressions), k-occurrence regular expressions and later work on schema inference. We acknowledge the grant 4938/4-1 by the Deutsche Forschungsgemeinschaft (DFG).

6. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.

[3] G. Bagan, A. Bonifati, and B. Groz. A trichotomy for regular simple path queries on graphs. In *PODS*, pages 261–272, 2013.

[4] A. Bielefeldt, J. Gonsior, and M. Krötzsch. Practical linked data access via SPARQL: the case of wikidata. In *LDOW@WWW*, 2018.

[5] A. Bonifati, W. Martens, and T. Timm. An analytical study of large SPARQL query logs. *PVLDB*, 11(2):149–161, 2017.

[6] A. Bonifati, W. Martens, and T. Timm. An analytical study of large SPARQL query logs. *The VLDB Journal*, 2019. Full version of [5], to appear.

[7] A. Bonifati, W. Martens, and T. Timm. Navigating the maze of wikidata query logs. In *WWW*, pages 127–138, 2019.

[8] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. A graphical query language supporting recursion. In *SIGMOD Conference*, pages 323–330, 1987.

[9] F. V. Fomin, D. Lokshtanov, F. Panolan, and S. Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *Journal of the ACM*, 63(4):29:1–29:60, 2016.

[10] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor. Cypher: An evolving query language for property graphs. In *SIGMOD Conference*, pages 1433–1445, 2018.

[11] G. Gottlob, G. Greco, N. Leone, and F. Scarcello. Hypertree decompositions: Questions and answers. In *PODS*, pages 57–74, 2016.

[12] A. S. LaPaugh and R. L. Rivest. The subgraph homeomorphism problem. *Journal of Computer and System Sciences*, 20(2):133 – 149, 1980.

[13] W. Martens, M. Niewerth, and T. Trautner. A trichotomy for regular trail queries. *CoRR*, abs/1903.00226, 2019.

[14] W. Martens and T. Trautner. Evaluation and enumeration problems for regular path queries. In *ICDT*, pages 19:1–19:21, 2018.

[15] W. Martens and T. Trautner. Dichotomies for evaluating simple regular path queries. *ACM Transactions on Database Systems*, 2019. Full version of [14], to appear.

[16] A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6):1235–1258, 1995.

[17] Neo4j. The neo4j developer manual v3.2. `https://neo4j.com/docs/developer-manual/3.2/`, 2017.

[18] Y. Perl and Y. Shiloach. Finding two disjoint paths between two pairs of vertices in a graph. *Journal of the ACM*, 25(1):1–9, 1978.

[19] J. Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

Technical Perspective: ϵKTELO: A Framework for Defining Differentially-Private Computations

Graham Cormode
University of Warwick, UK
G.Cormode@warwick.ac.uk

When was the last time that you wrote code to implement a join algorithm? Chances are, it was during an undergraduate database class – if at all. The wide availability of database management systems in all their manifestations (admitting a wide definition, to encompass performing look-ups in a spreadsheet) mean that we do not have to (re)implement common operations over and over again. This brings many advantages. We benefit from time savings, both in development time, and also in execution time: we can expect that optimized professional code will outperform our ad-hoc efforts. Moreover, we expect such code to be robust, and less prone to crashing on unexpected inputs. It should produce results that can be relied on to be correct, and handle errors gracefully.

Such "systematization" is a core methodology in computer science. Whenever we identify new areas where computation is needed, there is a sequence of steps that can be followed. First, we develop algorithms for special cases or particular operations. Over time, these move from proof-of-concept code into more reliable libraries and toolkits. From these, we abstract new collections of operations that together can be combined to address instances that might arise. Describing the sequence of steps to perform might initially be done via simple scripting or calls out from an existing high-level language, but over time may instead be expressed via a special purpose (and oft-times declarative) language, or through a graphical user interface. Eventually, we have a stand-alone system to describe tasks, which can be deployed by users who might otherwise lack the ability to code up the routines themselves.

Viewed through this lens, we can see many cases of systems emerging in computer science. The (relational) database management system is perhaps our default example. High-level languages themselves have also followed this path. Currently, machine learning tools are part way through this evolution: machine learning algorithms and libraries have been around for a while, but we are yet to achieve user-friendly systems that allow non-expert users to quickly and easily define complex machine learning pipelines.

The following paper by Zhang *et al.* talks in terms of a framework for a class of privacy-preserving computations over data. The artifact, ϵKTELO, represents an important step on the pathway to providing systems for such computations. It is not the first system in this domain. Frameworks such as PINQ [2], which extends the (non-private) LINQ framework, and Featherweight PINQ [1] are acknowledged as direct antecedents. ϵKTELO extends these by providing a different selection of primitives at higher levels of abstraction.

Computing under guarantees of privacy shares many similarities with secure computation. In particular, the mantra "Don't roll your own crypto" can equally well be transcribed as "Don't roll your own privacy". Subtle (and not so subtle) errors in defining and combining algorithms to protect the privacy of individuals motivate us to further automate and systematize the handling of private data. ϵKTELO assists by not only providing a broad set of tools for the most common operations on private data, but also by simplifying the analysis of the privacy properties of the resulting composition (captured by the sometimes inscrutable privacy parameter ϵ alluded to in the name). It more clearly separates the private from the public domain, and directs attention to steps which move information across this divide. The value of the framework is demonstrated not only by the ease with which a range of algorithms can be expressed, but also in the way it exposes new variants that can lead to improved utility.

By no means should we expect ϵKTELO to be the last word on this topic. There are a number of limitations that we can hope future work to overcome. Chief among these is the restriction to input that is modeled as a single table. Compare this to the DBMS, where we can SELECT, PROJECT and JOIN to our heart's content, all the while remaining fully within the world of relations. The ability to represent and compute by linking over different tables, and model networks of interactions, represents a pressing open problem for privacy computation that needs to be solved before systems can generalize to this case.

1. REFERENCES

[1] Hamid Ebadi and David Sands. Featherweight PINQ. *Journal of Privacy and Confidentiality*, 7(2):159–184, 2017.

[2] Frank McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD Conference*, 2009.

ϵKTELO: A Framework for Defining Differentially-Private Computations

Dan Zhang
U. of Massachusetts Amherst
dzhang@cs.umass.edu

Ryan McKenna
U. of Massachusetts Amherst
rmckenna@cs.umass.edu

Ios Kotsogiannis
Duke University
iosk@cs.duke.edu

George Bissias
U. of Massachusetts Amherst
gbiss@cs.umass.edu

Michael Hay
Colgate University
mhay@colgate.edu

Ashwin Machanavajjhala
Duke University
ashwin@cs.duke.edu

Gerome Miklau
U. of Massachusetts Amherst
miklau@cs.umass.edu

ABSTRACT

The adoption of differential privacy is growing but the complexity of designing private, efficient and accurate algorithms is still high. We propose a novel programming framework and system, ϵKTELO, for implementing both existing and new privacy algorithms. For the task of answering linear counting queries, we show that nearly all existing algorithms can be composed from operators, each conforming to one of a small number of operator classes. While past programming frameworks have helped to ensure the privacy of programs, the novelty of our framework is its significant support for authoring accurate and efficient (as well as private) programs.

We describe the design and architecture of the ϵKTELO system and show that ϵKTELO is expressive enough to describe many algorithms from the privacy literature. ϵKTELO allows for safer implementations through code reuse and allows both privacy novices and experts to more easily design new algorithms. We demonstrate the use of ϵKTELO by designing new algorithms offering state-of-the-art accuracy and runtime.

1. INTRODUCTION

As the collection of personal data has increased, many institutions face an urgent need for reliable privacy protection mechanisms. They must balance the need to protect individuals with demands to use the collected data—for new applications or modeling their users' behavior—or share it with external partners. Differential privacy [8, 9] is a rigorous privacy definition that offers a persuasive assurance to individuals, provable guarantees, and the ability to analyze the impact of combined releases of data. Informally, an algorithm satisfies differential privacy if its output does not change too much when any one record in the input database is added or removed.

The research community has actively investigated differential privacy and algorithms are known for a variety of tasks ranging from data exploration to query answering to machine learning. However, the adoption of differentially private techniques in real-world applications remains rare. We believe this is because implementing programs that provably satisfy privacy and ensure sufficient utility for a given task is still extremely challenging for non-experts in differential privacy. In fact, the few real world deployments of differential privacy—like OnTheMap [1, 15] (a U.S. Census Bureau data product), RAPPOR [11] (a Google Chrome extension), and Apple's private collection of emoji's and HealthKit data—have required teams of privacy experts to ensure that implementations meet the privacy standard and that they deliver acceptable utility. There are three important challenges in implementing and deploying differentially private algorithms.

The first and foremost challenge is the difficulty of designing utility-optimal algorithms: i.e., algorithms that can extract the maximal accuracy given a fixed "privacy budget." While there are a number of general-purpose differentially private algorithms, such as the Laplace Mechanism [8], they typically offer suboptimal accuracy if applied directly. A carefully designed algorithm can improve on general-purpose methods by an order of magnitude or more—without weakening privacy: accuracy is improved by careful engineering and sophisticated algorithm design.

One might hope for a single dominant algorithm for each task, but a recent empirical study [17] showed that the accuracy of existing algorithms is complex: no single algorithm delivers the best accuracy across the range of settings in which it may be deployed. The choice of the best algorithm may depend on the particular task, the available privacy budget, and properties of the input data such as its size and distribution. Therefore, to achieve state-of-the-art accuracy, a practitioner currently has to make a host of complex algorithm choices, which may include choosing a low-level representation for the input data, translating their queries into that representation, choosing among available algorithms, and setting parameters. The best choices will vary for different input data and different analysis tasks.

The second challenge is that the tasks in which practitioners are interested are diverse and may differ from those considered in the literature. Hence, existing algorithms need to be adapted to new application settings, a non-trivial task. For instance, techniques used by modern privacy algorithms include optimizing error over multiple queries by identifying common sub-expressions, obtaining noisy counts from the data at different resolutions, and using complex inference techniques to reconstruct answers to target queries

This is a minor revision of the work published in SIGMOD June 10–15, 2018, Houston, TX, USA.
https://doi.org/10.1145/3183713.3196921

from noisy, inconsistent and incomplete measurement queries. But different algorithms use different specialized operators for these sub-tasks, and it can be challenging to adapt them to new situations. Thus, designing utility-optimal algorithms requires significant expertise in a complex and rapidly-evolving research literature.

A third equally important challenge is that correctly implementing differentially private algorithms can be difficult. There are known examples of algorithm pseudocode in research papers not satisfying differential privacy as claimed. For instance, Zhang et al [42] showed that many variants of a privacy primitive called the Sparse Vector Technique do not satisfy differential privacy. Differential privacy can also be broken through incorrect implementations of sound algorithms. For example, Mironov [30] showed that standard implementations of basic algorithms like the Laplace Mechanism [8] can violate differential privacy because of their use of floating point arithmetic. Privacy-oriented programming frameworks such as PINQ [10, 29, 32], Fuzz [13], PrivInfer [6] and LightDP [38] help users implement programs whose privacy can be verified with relatively little human intervention. While they help to ensure the privacy criterion is met, they may impose their own restrictions and offer little or no support for designing utility-optimal programs.

Contributions

To address the aforementioned challenges, we propose ϵKTELO, a programming framework and system that aids programmers in developing differentially private programs with high utility. Programmers can use ϵKTELO to solve a core class of statistical tasks that involve answering *linear counting queries*. (This class of queries is defined in Sec. 2.) Tasks supported by ϵKTELO include releasing contingency tables, multi-dimensional histograms, answering OLAP and range queries, and implementing private machine learning algorithms.

In ϵKTELO, differentially private programs are described as *plans* over a high level library of *operators*. Each operator is an abstraction of a key subroutine from a state-of-the-art algorithm. Within ϵKTELO, these operators are organized based on their functionality into a small set of classes: transformation, querying, inference, query selection, and partition selection. These classes, and the operators within each class, are described in more detail in Sec. 3.2.

The design of ϵKTELO has the following characteristics:

Expressiveness ϵKTELO is designed to be expressive, meaning that a wide variety of state-of-the-art algorithms can be written succinctly as ϵKTELO plans. To ensure expressiveness, we carefully designed a foundational set of operator classes that cover features commonly used by leading differentially private algorithms. We illustrate the expressiveness of our operators by showing in Sec. 4 that the algorithms from the DPBench benchmark [17] can be readily re-implemented in ϵKTELO.

Privacy "for free" ϵKTELO is designed so that any plan written in ϵKTELO automatically satisfies differential privacy. The formal statement of this privacy property is in Sec. 3.3, the proof of which requires non-trivially extending the formal analysis of a past framework [10]. The privacy guarantee means that plan authors are not burdened with writing proofs for each algorithm they write. Furthermore plan authors do not need to think about how to calibrate the noise that is injected for privacy as this is handled automatically by ϵKTELO.

Reduced privacy verification effort Ensuring that an algorithm implementation satisfies differential privacy requires verifying that it matches the algorithm specification. The design of ϵKTELO reduces the amount of code that must be vetted in several ways. First, since an algorithm is expressed as a plan and all plans automatically satisfy differential privacy, the code to be vetted is solely the individual operators. Second, operators need to be vetted only once but may be reused across multiple algorithms. Finally, it is not necessary to vet every operator, but only those that are privacy-critical (as shown in Sec. 3.1, ϵKTELO mandates a clear distinction between privacy-critical and non-private operators). This means that verifying the privacy of an algorithm requires checking fewer lines of code. In Sec. 4, we compare the verification effort to vet the DPBench codebase [2] against the effort required to vet these algorithms when expressed as plans in ϵKTELO.

Transparency In ϵKTELO, since all algorithms are expressed in the same form—a plan, consisting of a sequence of operators where each operator is selected from a class—it is easy to compare algorithms and identify differences. In Sec. 4, we summarize the plan signatures of a number of state-of-the-art algorithms (pictured in Fig. 2). These plan signatures reveal similarities and common idioms in existing algorithms. These are difficult to discover from the research literature or through code inspection.

The modular structure of ϵKTELO creates opportunities for technical innovation. There are broadly two kinds of innovation. The first is performance improvements that can be embedded directly into the framework, for example by improving the implementation of a key operator or designing efficient implementations of essential data structures. The second is technical innovation that arises from *using* ϵKTELO, for example, by using the framework to design new algorithms. In this paper we present innovations of both kinds.

Improved Scalability Many of the operators in ϵKTELO represent data and queries using matrices. Performing matrix operations can become a performance bottleneck. In Sec. 5.1, we present a specialized matrix representation that avoids the expensive materialization of matrix objects. This *implicit matrix* representation did not appear in the first version of the ϵKTELO framework [40] but is described in the extended version [39].

A vitally important but computationally expensive operator in many plans is inference. Inference is used to combine a collection of noisy measurements into a consistent estimate of the private data. In Sec. 5.2, we introduce a general-purpose, efficient and scalable inference engine, which exploits implicit representations, and subsumes customized inference subroutines from the literature.

Algorithm innovation enabled by ϵKTELO Because ϵKTELO plans are composed of operators, improving existing algorithms and implementing new algorithms becomes much easier: operators can be combined in new ways, for example, by substituting one instance of an operator class for another. In Sec. 6, we present one example of algorithmic innovation. We describe a new algorithm that, when expressed as a plan in ϵKTELO, is similar to the MWEM algorithm [16] but with a few key operators replaced, which can lower error by as much as 8 times.

After providing background in Sec. 2, we describe the system fully in Sec. 3. We illustrate the expressiveness of ϵKTELO plans and the benefits of ϵKTELO in Sec. 4. Scalability innovations provided by ϵKTELO are presented in Sec. 5 while algorithmic innovations enabled by ϵKTELO are described in Sec. 6. We discuss related work and conclude in Secs. 7 and 8.

2. BACKGROUND

The input to ϵKTELO is a database instance of a single-relation schema $T(A_1, A_2, \ldots, A_\ell)$. Multi-relation schemas pose a number of challenges (please see discussion in Sec. 8). Each attribute A_i is assumed to be discrete (or suitably discretized). A *condition for-*

mula, ϕ, is a Boolean condition that can be evaluated on any tuple of T. We use $\phi(T)$ to denote the number of tuples in T for which ϕ is true. A number of operators in εKTELO answer linear queries over the table. A linear query is the linear combination of any finite set of condition counts:

DEFINITION 1 (LINEAR COUNTING QUERY (DECLARATIVE)). *A linear query q on T is defined by conditions $\phi_1 \dots \phi_k$ and coefficients $c_1 \dots c_k \in \mathbb{R}$ and returns $q(T) = c_1\phi_1(T) + \dots + c_k\phi_k(T)$.*

It is common to consider a vector representation of the database, denoted $\mathbf{x} = [x_1 \dots x_n]$, where x_i is equal to the number of tuples of type i for each possible tuple type in the relational domain of T. The size of this vector, n, is the product of the attribute domains. Then it follows that any linear counting query has an equivalent representation as a vector of n coefficients, and can be evaluated by taking a dot product with $\mathbf{x}$. Abusing notation slightly, let $\phi(i) = 1$ if ϕ evaluates to true for the tuple type i and 0 otherwise.

DEFINITION 2 (LINEAR COUNTING QUERY (VECTOR)). *For a linear query q defined by $\phi_1 \dots \phi_k$ and $c_1 \dots c_k$, its equivalent vector form is $\vec{q} = [q_1 \dots q_n]$ where $q_i = c_1\phi_1(i) + \dots + c_k\phi_k(i)$. The evaluation of the linear query is $\vec{q} \cdot \mathbf{x}$, where $\mathbf{x}$ is vector representation of T.*

In the sequel, we will use vectorized representations of the data frequently. We refer to the *domain* as the size of $\mathbf{x}$, the vectorized table. This vector is sometimes large and a number of methods for avoiding its materialization are discussed later.

Let T and T' denote two tables of the same schema, and let $T \oplus T' = (T - T') \cup (T' - T)$ denote the symmetric difference between them. We say that T and T' are neighbors if $|T \oplus T'| = 1$.

DEFINITION 3 (DIFFERENTIAL PRIVACY [8]). *A randomized algorithm $\mathcal{A}$ is ϵ-differentially private if for any two instances T, T' such that $|T \oplus T'| = 1$, and any subset of outputs $S \subseteq Range(\mathcal{A})$,*

$$Pr[\mathcal{A}(T) \in S] \leq \exp(\epsilon) \times Pr[\mathcal{A}(T') \in S]$$

Differentially private algorithms can be composed with each other, and other algorithms, using composition rules, such as sequential and parallel composition [29] and post-processing [9]. Let f be a function on tables that outputs real numbers. The *sensitivity* of the function is defined as: $max_{|T \oplus T'|=1}|f(T) - f(T')|$. The sensitivity of a function evaluated on a table (or vector) resulting from a sequence of transformations can be calculated from the *stability* of each transformation function:

DEFINITION 4 (STABILITY). *Let g be a transformation function that takes a data source (table or vector) as input and returns a new data source (of the same type) as output. For any pair of sources S and S' let $|S \oplus S'|$ denote the distance between sources. If the sources are both tables, then this distance is the size of the symmetric difference; if the sources are both vectors, then this distance is the L_1 norm. Then the stability of g is: $\max_{S,S':|S \oplus S'|=1} |g(S) \oplus g(S')|$.*

3. EKTELO

In this section, we describe the essential features of εKTELO: its system architecture, its operator-based programming framework, and its guarantee that every program written in the framework satisfies differential privacy.

3.1 System Architecture

In εKTELO, the private input data source is encapsulated inside a *protected kernel*. The analyst writes a differentially private program in an unprotected *client* space. Access to the protected data

Algorithm 1 εKTELO CDF Estimator

```
1:  D ← Protected(source_uri, ε)                        ▷ Initialize εKTELO
2:  D ← Where(D, sex == 'M' AND age ∈ [30, 39])          ▷ Transform
3:  D ← Select(salary)                                  ▷ Transform
4:  x ← T-Vectorize(D)                                  ▷ Transform
5:  P ← AHPartition(x, ε/2)                             ▷ Partition Selection
6:  x̄ ← V-ReduceByPartition (x, P)                      ▷ Transform
7:  M ← Identity(|x̄|)                                   ▷ Query Selection
8:  y ← VecLaplace(x̄, M, ε/2)                           ▷ Query
9:  x̂ ← NNLS(P, y)                                      ▷ Inference
10: W_pre ← Prefix(|x|)                                 ▷ Query Selection
11: return W_pre · x̂                                    ▷ Output
```

source is mediated by the protected kernel through a set of *operators*. The distinction between the client space and the protected kernel is a fundamental one in εKTELO. It allows analysts to write *plans* (of differentially private programs) that consist of operator calls to the data source embedded in otherwise arbitrary code (which may include conditionals, loops, recursion, etc.). εKTELO supports operators of three types, based on their interaction with the protected kernel. The first type is a **Private** operator, which requests that the protected kernel perform some action on the private data (e.g., a transformation) but receives only an acknowledgment that the operation has been performed. The second type is a **Private→Public** operator, which returns information about the private data outside the firewall (e.g., a differentially private measurement) and thus consumes privacy budget. The last type is a **Public** operator, which does not interact with the protected kernel or the protected data source at all and can be executed entirely in client space. We illustrate the operators supported by εKTELO in Sec. 3.2.

The protected kernel is initialized by specifying a single protected data object—an input table T—and a global privacy budget, ϵ. Note that requests for data transformations may cause the protected kernel to derive additional data sources (whose lineage is tracked to ensure correct privacy semantics).

We designed εKTELO to be extensible through the addition of new operators. The effort required depends on the operator type: Public operators can be added at will; Private operators must register their *stability* (Sec. 2); Private→Public operators must be vetted by a privacy engineer to ensure that they satisfy differential privacy (note, that εKTELO is responsible for appropriately calibrating the privacy budget when such operators are applied to derived data sources).

3.2 Operator Framework

In εKTELO, differentially private algorithms are described using *plans* composed over a rich library of *operators*. Most of the plans described in this paper are linear sequences of operators, but εKTELO also supports plans with iteration (as in plan #7, to be presented in Fig. 2), recursion, and branching. Operators supported by εKTELO perform a well defined task and typically capture a key algorithm design idea from the state-of-the-art. Each operator belongs to one of five *operator classes* based on its input-output specification. These are: (a) transformation, (b) query, (c) inference, (d) query selection, and (e) partition selection. All the operators supported by εKTELO are listed in Fig. 1 grouped by operator class and color coded by their type **Private**, **Private→Public** or **Public**. We next describe an example εKTELO plan and use it to introduce the different operator classes.

Algorithm 1 shows the pseudocode for a plan authored in εKTELO, which takes as input a table D with schema [Age, Gender, Salary] and returns the differentially private estimate of the empirical cumulative distribution function (CDF) of the Salary attribute, for males in their 30's. The plan is fairly sophisticated and works in multiple steps. The plan uses *transformation* operators on the in-

Transform	
TV	T-Vectorize
TW	T-Where
TPR	T-Project
TP	V-SplitByPartition
TR	V-ReduceByPartition

Inference	
LS	Least squares
NLS	Nneg Least squares
MW	Mult Weights
HR	Thresholding

Partition selection	
PA	AHPpartition
PG	Grid
PD	Dawa
PW	Workload-based
PS	Stripe(attr)
PM	Marginal(attr)
PU	UniformPartition

Query	
LM	Vector Laplace

Query selection	
SI	Identity
ST	Total
SP	Privelet
SH2	H2
SHB	HB
SG	Greedy-H
SU	UniformGrid
SA	AdaptiveGrids
SQ	Quadtree
SHD	HDMM
SS	Stripe(attr)
SW	Worst-approx
SPB	PrivBayes select

Figure 1: The operators currently implemented in ϵktelo. Private operators are red, Private→Public operators are orange, and Public operators are green.

put table D to filter out records that do not correspond to males in their 30's (Line 2), and to select only the salary attribute (Line 3). Then it uses another transformation operator to construct a vector of counts $\mathbf{x}$ that contains one entry for each value of salary. $\mathbf{x}[i]$ represents the number of rows in the input (in this case males in their 30's) with salary equal to i. All transformation operators are Private operators as they change the private database and do not return anything outside the protected kernel.

Before adding noise to this histogram, the plan uses a *partition selection* operator, AHPartition (Line 5). Operators in this class choose a partition $\mathbf{P}$ of the data vector $\mathbf{x}$ (or more generally, a mapping to a lower dimensional space) which is later used to transform $\mathbf{x}$. AHPartition is a key subroutine in AHP [43], which was shown to have state-of-the-art performance for histogram estimation [17]. AHPartition uses the data vector to identify a partition of the counts in $\mathbf{x}$ such that counts within a partition group are close. Since AHPartition uses the private input, it is a Private→Public operator and thus consumes privacy budget (in this case $\epsilon/2$). Fig. 1 shows other examples of partition selection operators that are data independent, and hence are Public operators.

Next the plan uses V-ReduceByPartition (Line 6), another transformation operator, which applies on $\mathbf{x}$ the partition $\mathbf{P}$ computed by AHPartition. This results in a new reduced vector $\bar{\mathbf{x}}$ that contains one entry for each partition group in $\mathbf{P}$ and the entry is computed by adding up counts within each group.

The rest of the plan follows the "select-measure-reconstruct" paradigm, an approach exemplified by the matrix mechanism [24, 27] and used in several state-of-the-art algorithms [17]. In this paradigm, in order to answer a workload of queries, the algorithm first *selects* a different set of strategy queries, *measures* them using the Laplace mechanism (with noise calibrated to the sensitivity of the strategy), and lastly *reconstructs* answers to the original workload of queries from the noisy measurements using inference algorithms. The careful selection of a low sensitivity strategy can often lead to much more accurate answers than directly answering the workload.

In our CDF estimation problem, the workload corresponds to a set of prefix queries of the form "# people with salary $< i$". Rather than asking these queries, the plan chooses the strategy of "identity," which corresponds to queries of the form "# people with salary $= i$". In Algorithm 1, this is captured by the Identity operator (Line 7). This operator is a *query selection* operator. Such operators specify a set of measurement queries $\mathbf{M}$, encoded in matrix form to be applied to the data vector. The Identity operator returns the identity matrix, which corresponds to querying all the entries in $\bar{\mathbf{x}}$ (since $\mathbf{M} \cdot \bar{\mathbf{x}} = \bar{\mathbf{x}}$). In general, query selection operators do not answer any query, but rather specify which queries should be estimated. (This is analogous to how partition selection operators only select a partition but do not apply it.) Most query selection operators are data independent and thus are Public, while some use the data to select the query strategy, and hence are Private→Public.

Next, the plan performs the measurement step using the Vector Laplace operator, which returns differentially private answers to all the queries in $\mathbf{M}$. First, it automatically calculates the sensitivity of the vectorized queries – which depends on all upstream data transformations – and then adds noise via the standard Laplace mechanism (Line 8). This operator consumes the remainder of the privacy budget. Query operators like Vector Laplace return a noisy measurement from the data, and by definition are differentially private algorithms that expend privacy budget. They are Private→Public.

So far the plan has computed an estimated histogram of partition group counts $\mathbf{y}$, but now it must *reconstruct* the empirical CDF on the original salary domain. From the noisy counts on the reduced domain $\mathbf{y}$, the plan infers non-negative counts in the original vector space of $\mathbf{x}$ by invoking an *inference* operator NNLS (non-negative least squares) (Line 9). NNLS($\mathbf{P}, \mathbf{y}$) finds a solution, $\hat{\mathbf{x}}$, to the problem $\mathbf{P}\hat{\mathbf{x}} = \mathbf{y}$, such that all entries of $\hat{\mathbf{x}}$ are non-negative. Inference operators never access the protected data and thus can be safely run outside the protected kernel. All inference operators are Public.

Lastly, the plan constructs the set of queries, $\mathbf{W_{pre}}$, needed to compute the empirical CDF (a lower triangular $k \times k$ matrix representing prefix sums) by calling the query selection operator Prefix(k) (Line 10), and returns the output (Line 11).

3.3 Privacy Guarantee

In this section, we state the privacy guarantee offered by ϵktelo. Informally, ϵktelo ensures that if the protected kernel is initialized with a source database T and a privacy budget ϵ, then any plan (chosen by the client) satisfies ϵ-differential privacy. ϵktelo ensures privacy by tracking the privacy budget consumed by each operator call. The amount spent depends on the operator (e.g., Public operators consume nothing) and on what operations came before it (e.g., transformations). If at any point, the privacy budget is exhausted, any subsequent call to an operator that requires budget (i.e., a Private→Public operator) will throw an exception.

In the proof of privacy, we model a client's plan as an arbitrary and possibly infinite sequence of operator calls, where each call may be adaptively chosen based on the results of earlier calls. Thus we can think of the plan as a random process, where the randomness comes from the randomness of the operators (though the client code could also be randomized). The length of the plan, measured by the number of operator calls, can vary but we can nevertheless consider the set of (partial) plans of length k for any k. (A shorter plan can be extended with "no op" calls and for a longer plan, we consider the prefix of its first k operators.) This allows us to define a probability distribution over outcomes after k operations where an outcome is a particular length k plan, denoted $p_1 p_2 \dots p_k$, and the outputs received from executing that plan, denoted $o_1 o_2 \dots o_k$.

Theorem 3.1 (Privacy of ϵktelo plans). *Let T, T' be any two neighboring instances. For all $k \in \mathbb{N}^+$,*

$$P(\textit{Plan is } p_1 p_2 \dots p_k \textit{ with outputs } o_1 o_2 \dots o_k \mid \textit{Ektelo init. with } (T, \epsilon))$$
$$\leq e^{\epsilon} P(\textit{Plan is } p_1 p_2 \dots p_k \textit{ with outputs } o_1 o_2 \dots o_k \mid \textit{Ektelo init. with } (T', \epsilon))$$

The proof of Theorem 3.1 extends the proof in [10] to support the V-SplitByPartition operator. While ϵktelo ensures differential privacy, private information could be leaked via side-channel attacks (e.g., timing attacks) [14]. Privacy engineers who design operators are responsible for protecting against such attacks; an analysis of this issue is beyond the scope of this paper.

ID	Cite	Algorithm name	Plan signature							
1	Dwork et al. 2006	Identity	SI	LM						
2	Xiao et al. 2010	Privelet	SP	LM	LS					
3	Hay et al. 2010	Hierarchical (H2)	SH2	LM	LS					
4	Qardaji et al. 2013	Hierarchical Opt (HB)	SHB	LM	LS					
5	Li et al. 2014	Greedy-H	SG	LM	LS					
6	-	Uniform	ST	LM	LS					
7	Hardt et al. 2012	MWEM	I:(	SW	LM	MW	)			
8	Zhang et al. 2014	AHP	PA	TR	SI	LM	LS			
9	Li et al. 2014	DAWA	PD	TR	SG	LM	LS			
10	Cormode et al. 2012	Quadtree	SQ	LM	LS					
11	Qardaji et al. 2013	UniformGrid	SU	LM	LS					
12	Qardaji et al. 2013	AdaptiveGrid	SU	LM	LS	PU	TP[	SA	LM]	LS
13	NEW	MWEM variant b	I:(	SW	SH2	LM	MW	)		
14	NEW	MWEM variant c	I:(	SW	LM	NLS	)			
15	NEW	MWEM variant d	I:(	SW	SH2	LM	NLS	)		

Figure 2: The high-level signatures of a subset of plans implemented in ϵKTELO (referenced by ID). All plans begin with a vectorize transformation, omitted for readability. We also omit parameters of operators, including ϵ budget shares. `I`(*subplan*) refers to iteration of a subplan and `TP`[*subplan*] means that *subplan* is executed on each partition produced by `TP`.

4. EKTELO BENEFITS

ϵKTELO provides a number of benefits for the authors of differentially private programs, including code reuse, transparency, expressiveness, and a significant reduction in privacy verification effort.

We illustrate these benefits by reporting on our experience reimplementing state-of-the-art algorithms as ϵKTELO plans. We examined 12 algorithms for answering low-dimensional counting queries that were deemed competitive in a recent benchmark study [17]. The process of re-implementing this seemingly diverse set of algorithms consisted of identifying and isolating key subroutines and translating them into operators.

The prototype implementation of ϵKTELO, including all algorithms used in experiments, consists of 7.9k lines of Python code. The framework itself makes up 25% while operator implementations make up 46% and 15% are tests and examples provided for users. The remaining 14% are definitions of plans used in our experiments, showing that once operators are defined, plan definitions are relatively simple. In fact, plans can be described and compared at a high level by looking at *plan signatures*. Fig. 2 describes the 12 re-implemented algorithms (numbered 1 through 12) using the abbreviations for operators introduced in Fig. 1. We use these plan signatures to highlight the following benefits:

Code reuse Operations that are common to many plans can be implemented once and reused across ϵKTELO programs. For example, once reformulated in ϵKTELO, nearly all the algorithms in Fig. 2 use the Vector Laplace operator (LM) and least squares inference (LS). This benefits plan authors since it simplifies and accelerates their ability to write new differentially private algorithms without having to reimplement sophisticated and privacy critical operators. In addition, any improvements to these operators will be inherited by all the plans. We show such an example in Sec. 5.2.

Algorithm transparency By rewriting algorithms as plans, ϵKTELO makes explicit the typical high-level patterns that reflect design idioms of algorithms in literature. For example, plans 2, 3, 4, 5, 6, 10, and 11 all share a common operator sequence: Query selection, Query (LM), and Inference (LS); they differ only in the specifics of their Query selection method. Moreover, ϵKTELO plans help clarify the distinctive components of complex state-of-the-art algorithms. For instance, AHP and DAWA (plans 8 and 9 in Fig. 2) have the same structure but differ only in two operators: partition selection and query selection.

Reduced privacy verification effort Code reuse also reduces the number of critical operators that must be carefully vetted. To verify privacy, the only operators that require careful vetting are those that consume the privacy budget, which are the Private→Public operators in Fig. 1. These are: Vector Laplace, the partition selection operators for both DAWA [23] and AHP [43], a query selection operator used by PrivBayes [41], and a query selection operator used by the MWEM [16] algorithm, which privately derives the worst-approximated workload query. In contrast, for the DPBench code base, the entire code has to be vetted to audit the use and management of the privacy budget. As a result, fewer lines of code must be verified as correct. For example, to verify the QuadTree algorithm in the DPBench codebase requires checking 163 lines of code. However, with ϵKTELO, this only requires vetting the 30-line Vector Laplace operator. And, furthermore, by vetting just this one operator, we have effectively vetted 10 of the 18 algorithms in Fig. 2, since the only privacy sensitive operator these algorithms use is Vector Laplace.). Considering all the DPBench algorithms in Fig. 2, algorithms 1-12, verifying the DPBench implementation requires checking a total of 1837 lines of code while vetting all the privacy-critical operators in ϵKTELO requires checking only 517 lines of code.

5. SCALABILITY INNOVATIONS

In this section we describe a number of innovations that enable key operations in ϵKTELO to scale to larger problem instances.

5.1 Implicit matrix representations

Matrices and operations on matrices are central to the implementation of ϵKTELO operators but can become a performance bottleneck. In an extension [39] to the original version [40] of ϵKTELO, we describe a set of specialized matrix representation techniques, based on the *implicit* definition of matrices, which allows for performance improvements and greater scalability as the size of the data vector grows.

Matrices are used to represent three different objects in ϵKTELO: sets of workload queries, sets of measurement queries, and partitions of the domain. In all cases the matrices contain one column for each element of a corresponding data vector. In the case of both workload and measurement matrices, rows represent linear queries. The number of rows in a workload or measurement matrix is often as large, or larger, than n, the number of elements in the data vector. Partition matrices have at most n rows, but may still be large. For plans operating on large data vectors, where n approaches the size of memory, these matrices, in standard form, are infeasible to represent in memory and operate on.

While distributed computation would be one solution, we find that we can remain with a main memory implementation by using a set of performance enhancements based on two key observations. First, the large matrix objects used in plans tend to possess structure that can be exploited to represent them very concisely. Second, the plan implementations in ϵKTELO use a relatively small set of basic operations on these matrix objects (e.g. matrix-vector product, matrix multiplication, transpose, absolute value). Together, these observations allow matrices to be represented and operated on *implicitly*, which results in significant performance improvements.

As an example of an implicit matrix, recall the Prefix workload, $\mathbf{W}_{\mathbf{pre}}$, an encoding of an empirical CDF, which was used in the example plan (Algorithm 1). In *explicit* form, the prefix workload has n^2 entries and is defined as a lower-triangular matrix containing 1's. Note that a *sparse* representation of $\mathbf{W}_{\mathbf{pre}}$ would store (a list of) only the nonzero elements of this matrix, but the space complexity remains $O(n^2)$. Thus, the time complexity of computing matrix-

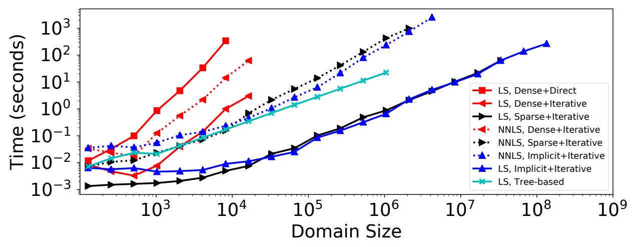


Figure 3: For a given computation time, the proposed iterative and implicit inference methods permit scaling to data vector sizes as much as 1000× larger than previous techniques that use direct approaches and dense matrices.

vector products using an explicit or sparse representation is $O(n^2)$. However, the Prefix matrix can be completely specified by a single parameter, n, which is the only state stored for the implicit version of $\mathbf{W}_{\mathbf{pre}}$. Further, we can evaluate the matrix-vector product $\mathbf{y} = \mathbf{W}_{\mathbf{pre}}\mathbf{x}$ using a simple one-pass algorithm over $\mathbf{x}$. Thus, the *implicit* Prefix workload representation achieves $O(1)$ space complexity and $O(n)$ time complexity for computing matrix-vector products.

While the original version of ϵKTELO exploited sparse matrix representations for some objects, we improve and extend our matrix representations to include implicit matrices in [39]. We describe a set of core implicit matrices, operations to combine them to form new implicit matrices, along with implementations of key operations on matrices, demonstrating significant performance improvements for ϵKTELO plans. One of the most important operations on implicit matrices is inference, which we discuss next.

5.2 Scalable and general inference

Inference is a fundamental operator that can improve error with no cost to privacy and, accordingly, we see that it appears in almost every algorithm re-implemented in ϵKTELO (as shown in Fig. 2). But inference can be a costly operation. Recall that the input to inference is a measurement matrix, denoted by $\mathbf{M}$, containing m queries defined over a data vector of size n, along with the list of noisy measurement answers $\mathbf{y}$. The most common inference method in existing algorithms is based on minimizing squared error:

Definition 5 (Ordinary least squares (LS)).

$$\hat{\mathbf{x}} = \underset{x \in \mathbb{R}^n}{\arg\min} \, \|\mathbf{M}\mathbf{x} - \mathbf{y}\|_2 \qquad (1)$$

The least squares solution (Eq. (1)) is given by the solution to the normal equations $\mathbf{M}^T\mathbf{M}\hat{\mathbf{x}} = \mathbf{M}^T\mathbf{y}$. Assuming $\mathbf{M}^T\mathbf{M}$ is invertible, then the solution is unique and can be expressed as $\hat{\mathbf{x}} = (\mathbf{M}^T\mathbf{M})^{-1}\mathbf{M}^T\mathbf{y}$. Matrix inversion is often avoided in favor of matrix factorizations of $\mathbf{M}$, but these methods, which we will refer to as *direct*, have time complexity cubic in the domain size, making it unacceptable when n is greater than about 5000.

Algorithms in prior work [18, 33, 34, 37] have performed least squares inference on large domains by restricting the selection of queries, namely to those representing a set of hierarchical queries. For this special case, inference can be performed in time linear in the domain size, avoiding the explicit matrix representation of the queries. We avoid this approach in ϵKTELO because it means that a custom inference method may be required for each query selection operation, and because it limits the measurement sets that can be used. In addition, hierarchical methods only work for least squares but not other inference methods, such as least squares with non-negativity constraints (NNLS).

An alternative to the *direct* implementation of least squares inference uses an *iterative* gradient-based method, which solves the normal equations by repeatedly computing matrix-vector products $\mathbf{M}\mathbf{v}$ and $\mathbf{M}^T\mathbf{v}$ until convergence. The time complexity of these methods is $O(kn^2)$ for dense matrix representations where k is the number of iterations. We use a well-known iterative method, LSMR [12], and empirically we observe LSMR to converge in far fewer than n iterations when $\mathbf{M}$ is well-conditioned, and thus we expect $k << n$.

Table 1: For three new algorithms, (b), (c), and (d), the multiplicative factors by which error is improved, presented as (min, mean, max) over datasets. For runtime, the mean is shown, normalized to the runtime of standard MWEM. (1D, n=4096, W=RandomRange(1000), $\epsilon = 0.1$)

	MWEM Variants		ERROR IMPROVEMENT			RUNTIME
	Measure Selection	**Inference**	min	mean	max	mean
(a)	worst-approx	MW	1	1	1	1
(b)	worst-approx + H2	MW	1.03	2.80	7.93	354.9
(c)	worst-approx	NNLS, known total	0.78	1.08	1.54	1.0
(d)	worst-approx + H2	NNLS, known total	0.89	2.64	8.13	9.0

The benefits of iterative inference methods are amplified when the underlying matrix representation is implicit. Letting $Time(\mathbf{M})$ denote the time complexity of evaluating a matrix-vector product with $\mathbf{M}$, the time complexity of least squares inference is $O(k \cdot Time(\mathbf{M}))$ where k is the number of iterations. For implicit matrices, $Time(\mathbf{M})$ is often $O(n)$, resulting in a very favorable $O(kn)$ time complexity for inference. Iterative approaches, using implicit matrices, are also well-suited to the other inference methods in ϵKTELO: least squares with non-negativity constraints (NNLS) and multiplicative weights (MW).

We compare the performance of the above approaches for a fixed measured query set consisting of binary hierarchical measurements [18]. Fig. 3 shows that using sparse matrices and iterative methods allows inference to scale to data vectors consisting of millions of counts on a single machine in less than a minute. The use of implicit matrices permits additional scale-up for both LeastSquares and NNLS. We also compare against the custom inference method introduced by Hay et al., denoted 'Tree-based' in the figure. It is an algorithm that is logically equivalent to LeastSquares but specialized for hierarchically structured measurements. The general-purpose LeastSquares implementation is able to scale to much larger domains.

Importantly, the combination of general implicit matrix construction techniques with iterative inference results in flexible inference capabilities for plan authors. With relative freedom, authors can construct measurement matrices, or combine measurements from different parts of a plan, and apply a single generic inference operator, which will run efficiently.

6. ALGORITHMIC INNOVATIONS

The operator-based model of ϵKTELO enables novel improvements to algorithm design through (i) *operator inception*, in which a new operator is proposed for an operator class; (ii) *recombination*, where different operator instances are substituted for those in an existing plan to form a new plan; and (iii) *plan restructuring*, in which a plan is systematically restructured by applying a general design principle or heuristic rule. In the original version of this paper [40], we provide detailed examples of each of these innovation types. Below we provide a single example, improving the well-known MWEM algorithm, through *operator inception* and *recombination*.

The *Multiplicative Weights Exponential Mechanism* (MWEM) [16] algorithm computes answers to a given workload of linear queries. MWEM operates in a sequence of rounds, determined by an input parameter. In each round it selects the worst-approximated workload query with respect to its current estimate of the data, measures the selected query, and then uses the multiplicative weights update rule to refine its estimate of the data.

When viewed as a plan in ϵKTELO, a deficiency of MWEM becomes apparent. Its query selection operator selects a *single* query to measure in each round, whereas most query selection operators select a set of queries, each measuring disjoint partitions of the data. By the parallel composition property of differential privacy, measuring the entire set has the same privacy cost as asking any single query from the set. This means that MWEM could be measuring more than a single query per round (with no additional consumption of the privacy budget).

To exploit this opportunity, we designed a new query selection operator that adds to the worst-approximated query by attempting to build a binary hierarchical set of queries over the sequence of rounds of the algorithm. In round one, it adds any unit length queries that do not intersect with the selected query. In round two, it adds length two queries, and so on.

Adding more measurements to MWEM has an undesirable side effect on runtime, however. Because it measures a much larger number of queries across rounds of the algorithm and the runtime of multiplicative weights inference scales with the number of measured queries, inference can be considerably slower. Thus, we also use *recombination* to replace it with a version of least-squares with a non-negativity constraint (NNLS) and incorporate a high-confidence estimate of the total which is assumed by MWEM.

Using ϵKTELO, it was easy to describe three MWEM variants, which are shown in Fig. 2: an alternative query selection operator (PLAN #13) which augments selected measurements with hierarchical queries, an alternative inference operator (NNLS) (PLAN #14), and the addition of both alternative operators (PLAN #15).

Table 1 shows the experimental results over a collection of 10 datasets taken from [17]. The performance of the first variant on line (b) shows that the new query selection operator can significantly improve error: by a factor of 2.8 on average and by as much as a factor of 7.9. As explained above, it also has a considerable impact on performance because inference must operate on a larger set of queries: the slow down is a factor of more than 300. Line (c) shows that using the original MWEM query selection with NNLS inference has largely equivalent error and runtime to the original MWEM. However, combining augmented query selection with NNLS inference, shown on line (d), reduces runtime significantly while maintaining good accuracy: it is still slower than the original MWEM algorithm, but by only a factor of 9. The performance gains of NNLS inference over MW appear to be most pronounced when the number of measured queries is large. Overall, the algorithm variant (d) would likely be favored by users, who are typically willing to sacrifice efficiency for greater accuracy.

7. RELATED WORK

A number of languages and programming frameworks have been proposed to make it easier for users to write private programs [10, 29,32,35]. The *Privacy Integrated Queries* (PINQ) platform began this line of work and is an important foundation for ϵKTELO. We use the fundamentals of PINQ to ensure that plans implemented in ϵKTELO are differentially private. In particular, we adapt and extend a formal model of a subset of PINQ features, called Featherweight PINQ [10], to show that plans written using ϵKTELO operators satisfy differential privacy. Our extension adds support for the partition operator, a valuable operator for designing complex plans.

Additionally, there is a growing literature on formal verification tools that prove that an algorithm satisfies differential privacy [4,6, 13,38]. For instance, LightDP [38] is a simple imperative language in which differentially private programs can be written, allowing verification with little manual effort. LightDP's goal is orthogonal to that of ϵKTELO: it simplifies proofs of privacy, while ϵKTELO's goal is to simplify the design of algorithms that achieve high accuracy.

Concurrently with our work, Kellaris et al. [19] observe that algorithms for single-dimensional histogram tasks share subroutines that perform common functions.

The use of inference appears in many differentially private algorithms [3, 5, 7, 18, 22, 23, 25, 31, 36, 37, 43]. Proserpio et al. [31] propose a general-purpose inference engine based on MCMC that leverages properties of its operators to offset the otherwise high time/space costs of this form of inference. Our work is complementary in that we focus on a different kind of inference (based on least squares) in part because it is used, often implicitly, in many published algorithms.

A full treatment of automated plan optimization is an important future goal for ϵKTELO, however ϵKTELO could directly incorporate limited forms of automation already proposed in the literature. The matrix mechanism [25,27] formulates an optimization problem that corresponds to automated query selection in ϵKTELO. Other recent work [20, 26] considers the problem of data-dependent algorithm selection. These methods could be adapted to automatically select from a set of predefined plans in ϵKTELO.

8. CONCLUSIONS AND FUTURE WORK

We have described the design and implementation of ϵKTELO: an extensible programming framework and system for defining differentially private algorithms. Many state-of-the-art differentially private algorithms can be specified as ϵKTELO plans, consisting of sequences of operators, increasing code reuse and facilitating more transparent algorithm comparisons. Algorithms implemented in ϵKTELO are often faster and return more accurate answers.

ϵKTELO is extensible and we hope to expand the classes of tasks that can be supported. First, ϵKTELO is currently focused on statistical queries on a single table. Supporting more expressive aggregate queries, for example those expressible as SQL queries over multi-relational schemas, will require a number of extensions, including more complex transformations, advanced stability analysis, and improved inference [21].

Second, ϵKTELO currently relies on materializing the data vector in memory. Larger data vectors often occur with high-dimensional data, and while parallel computation is one possible solution, private algorithms may perform better when they are structured as a collection of private measurements over lower-dimensional projections of the original data. The ϵKTELO architecture is well-suited to perform transformation, selection, and measurement in such plans, and could adopt recently proposed methods [28] to perform "global" inference without materializing the full data vector.

Third, ϵKTELO provides a promising foundation for automatic optimization. Much like a relational optimizer, we envision adding a component that can explore a plan space implied by the collection of operators implemented in ϵKTELO. However, optimization here has dual objectives (accuracy and efficiency) and, in addition, it may be important to accommodate user-defined accuracy metrics. Further, the accuracy of some plans depends on the input data and may incur a privacy cost if it is used naively by the system during optimization.

Acknowledgments: Work supported by the National Science Foundation under grants 1253327, 1408982, 1409125, 1443014, 1421325, and 1409143; and by DARPA under contract N66001-15-C-4067.

9. REFERENCES

[1] https://onthemap.ces.census.gov/, 2010.

[2] https://github.com/dpcomp-org/dpcomp_core, 2016.

[3] G. Ács, C. Castelluccia, and R. Chen. Differentially private histogram publishing through lossy compression. In *ICDM*, pages 1–10, 2012.

[4] A. Albarghouthi and J. Hsu. Synthesizing coupling proofs of differential privacy. *Proc. ACM Program. Lang.*, (POPL), Dec. 2017.

[5] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency too: A holistic solution to contingency table release. In *PODS*, pages 273 – 282, 2007.

[6] G. Barthe, G. P. Farina, M. Gaboardi, E. J. G. Arias, A. Gordon, J. Hsu, and P.-Y. Strub. Differentially private bayesian programming. In *CCS*, pages 68–79, 2016.

[7] G. Cormode, M. Procopiuc, E. Shen, D. Srivastava, and T. Yu. Differentially private spatial decompositions. In *ICDE*, pages 20–31, 2012.

[8] C. Dwork, F. M. K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.

[9] C. Dwork and A. Roth. *The Algorithmic Foundations of Differential Privacy*. Foundations and Trends in Theoretical Computer Science, 2014.

[10] H. Ebadi and D. Sands. Featherweight pinq. *JPC*, 7(2), 2017.

[11] Ú. Erlingsson, V. Pihur, and A. Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *CCS*, 2014.

[12] D. C.-L. Fong and M. Saunders. LSMR: An iterative algorithm for sparse least-squares problems. *SIAM J. Sci. Comput.*, 33(5):2950–2971, Oct. 2011.

[13] M. Gaboardi, A. Haeberlen, J. Hsu, A. Narayan, and B. C. Pierce. Linear dependent types for differential privacy. In *POPL*, pages 357–370, 2013.

[14] A. Haeberlen, B. C. Pierce, and A. Narayan. Differential privacy under fire. In *USENIX Conference on Security*, 2011.

[15] S. Haney, A. Machanavajjhala, J. Abowd, M. Graham, M. Kutzbach, and L. Vilhuber. Utility cost of formal privacy for releasing national employer-employee statistics. In *SIGMOD*, 2017.

[16] M. Hardt, K. Ligett, and F. McSherry. A simple and practical algorithm for differentially private data release. In *NIPS*, 2012.

[17] M. Hay, A. Machanavajjhala, G. Miklau, Y. Chen, and D. Zhang. Principled evaluation of differentially private algorithms using dpbench. In *SIGMOD*, 2016.

[18] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *PVLDB*, 2010.

[19] G. Kellaris, S. Papadopoulos, and D. Papadias. Differentially private histograms for range-sum queries: A modular approach. *arXiv*, 2015.

[20] I. Kotsogiannis, A. Machanavajjhala, M. Hay, and G. Miklau. Pythia: Data dependent differentially private algorithm selection. In *SIGMOD*, 2017.

[21] I. Kotsogiannis, Y. Tao, A. Machanavajjhala, G. Miklau, and M. Hay. Architecting a differentially private SQL engine. In *Conf. on Innovative Data Systems Research (CIDR)*, 2019.

[22] J. Lee, Y. Wang, and D. Kifer. Maximum likelihood postprocessing for differential privacy under consistency constraints. In *KDD*, 2015.

[23] C. Li, M. Hay, and G. Miklau. A data- and workload-aware algorithm for range queries under differential privacy. *PVLDB*, 2014.

[24] C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor. Optimizing linear counting queries under differential privacy. In *PODS*, pages 123–134, 2010.

[25] C. Li, G. Miklau, M. Hay, A. McGregor, and V. Rastogi. The matrix mechanism: optimizing linear counting queries under differential privacy. *The VLDB Journal*, pages 1–25, 2015.

[26] J. Liu and K. Talwar. Private selection from private candidates. *CoRR*, abs/1811.07971, 2018.

[27] R. McKenna, G. Miklau, M. Hay, and A. Machanavajjhala. Optimizing error of high-dimensional statistical queries under differential privacy. *PVLDB*, 11(10), 2018.

[28] R. McKenna, D. Sheldon, and G. Miklau. Graphical-model based estimation and inference for differential privacy. In *ICML*, 2019.

[29] F. D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD*, pages 19–30, 2009.

[30] I. Mironov. On significance of the least significant bits for differential privacy. In *CCS*, 2012.

[31] D. Proserpio, S. Goldberg, and F. McSherry. A workflow for differentially-private graph synthesis. In *Workshop on online social networks*, 2012.

[32] D. Proserpio, S. Goldberg, and F. McSherry. Calibrating data to sensitivity in private data analysis: A platform for differentially-private analysis of weighted datasets. *Proc. VLDB Endow.*, 7(8):637–648, Apr. 2014.

[33] W. Qardaji, W. Yang, and N. Li. Differentially private grids for geospatial data. In *ICDE*, pages 757–768. IEEE, 2013.

[34] W. Qardaji, W. Yang, and N. Li. Understanding hierarchical methods for differentially private histograms. *PVLDB*, 2013.

[35] I. Roy, S. T. V. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for mapreduce. In *NSDI*, 2010.

[36] O. Williams and F. McSherry. Probabilistic Inference and Differential Privacy. *NIPS*, pages 2451–2459, 2010.

[37] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. In *ICDE*, pages 225–236, 2010.

[38] D. Zhang and D. Kifer. Lightdp: Towards automating differential privacy proofs. In *POPL*, pages 888–901, 2017.

[39] D. Zhang, R. McKenna, I. Kotsogiannis, G. Bissias, M. Hay, A. Machanavajjhala, and G. Miklau. Ektelo: A framework for defining differentially-private computations. https://arxiv.org/abs/1808.03555v3.

[40] D. Zhang, R. McKenna, I. Kotsogiannis, M. Hay, A. Machanavajjhala, and G. Miklau. Ektelo: A framework for defining differentially-private computations. In *ACM Conference on Management of Data (SIGMOD)*, pages 115–130, 2018.

[41] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. PrivBayes: Private data release via Bayesian networks. *TODS*, 42, 2017.

[42] J. Zhang, X. Xiao, and X. Xie. Privtree: A differentially private algorithm for hierarchical decompositions. In *SIGMOD*, 2016.

[43] X. Zhang, R. Chen, J. Xu, X. Meng, and Y. Xie. Towards accurate histogram publication under differential privacy. In *SDM*, 2014.

Technical Perspective: Entity Matching with Quality and Error Guarantees

Benny Kimelfeld
Technion, Israel
bennyk@cs.technion.ac.il

Wim Martens
University of Bayreuth, Germany
wim.martens@uni-bayreuth.de

The challenge of *entity matching* is that of identifying when different data items (often referred to as *records* or *mentions*) refer to the same real-life entity. Popular instantiations of this problem include *deduplication*, where the items are database records that include duplicate representations of the same entity (e.g., duplicate profiles in a social network) [2], *record linkage*, where the items come from different data sources that mention overlapping sets of entities (e.g., the profiles of two social networks) [5], and *schema matching*, where the items are attributes of different database schemas that intersect on their domain of interest (e.g., the database schemas of different social networks) [6].

Common techniques for entity matching share various conceptual steps. First, *blocking* breaks the problem into considerably smaller subsets (blocks) of item pairs that have a reasonable chance to be matched, in order to reduce the quadratic number of needed comparisons. On each remaining pair to consider, a collection of *similarity functions* is applied to construct a vector of similarity scores. Next, a *classifier* transforms the vector into a decision: *match* or *non-match*. This classifier is typically built using supervised machine learning, where training is done over entity pairs labeled positively and negatively. Often, classification is complemented by a clustering algorithm if the matching is required to be transitive (i.e., if a profile matches a second profile, which matches a third profile, then the first must also match the third) [3].

There are other techniques for entity matching, including rule-based linking, and entity resolution via probabilistic inference. However, the field is generally short of fundamental guiding theory [4]. The paper "Entity Matching with Active Monotone Classification" [7] by Yufei Tao is a beautiful piece of work that proposes a principled approach to learn the aforementioned classification task over the vector of similarity scores, and more importantly, to reason about the theoretical bounds and the optimality of learning strategies.

The crux of the paper's development is to adopt an assumption that is very reasonable in the specific use case of the classifier: if *every* similarity function thinks that one pair is a better match than another, and if the latter is classified as a match, then the former should also be classified as a match. A classifier that features this behavior is called *monotone*, and the paper studies the learnability of monotone classifiers.

It is of course possible that no monotone classifier exists that is perfectly correct, i.e., perfectly separates matches from non-matches. Therefore, the author focuses on tradeoffs between the number of errors a classifier makes and the number of pairs that need to be *probed* (checked if they are a match or not).

The main algorithm in the paper, *random probe with elimination* (RPE) has several properties that could make it quite appealing to practitioners. It just consists of six lines of code and is extremely simple. Nevertheless, the author shows that it has favorable theoretical guarantees: it ensures an asymptotically optimal tradeoff between the number of probes and and the number of misclassified matches. Furthermore, as the algorithm is based on random sampling, it is expected to scale quite well.

Yufei Tao's paper not only offers us a nice blend between theory and practice, it is also a nice blend between databases and machine learning, which fits perfectly in some of the main research perspectives for the Principles of Data Management field [1].

1. REFERENCES

[1] S. Abiteboul et al. Research directions for principles of data management (Dagstuhl perspectives workshop 16151). *Dagstuhl Manifestos*, 7(1):1–29, 2018.

[2] A. Arasu, C. Ré, and D. Suciu. Large-scale deduplication with constraints using dedupalog. In *ICDE*, pages 952–963. IEEE Computer Society, 2009.

[3] P. Christen. *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.* Data-Centric Systems and Applications. Springer, 2012.

[4] L. Getoor and A. Machanavajjhala. Entity resolution: Theory, practice & open challenges. *PVLDB*, 5(12):2018–2019, 2012.

[5] T. N. Herzog, F. J. Scheuren, and W. E. Winkler. *Data quality and record linkage techniques*. Springer, 2007.

[6] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[7] Y. Tao. Entity matching with active monotone classification. In *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, pages 49–62, 2018.

Entity Matching with Quality and Error Guarantees

Yufei Tao[*]
Chinese University of Hong Kong
Hong Kong
taoyf@cse.cuhk.edu.hk

ABSTRACT

Given two sets of entities X and Y, *entity matching* aims to decide whether x and y represent the same entity for each pair $(x, y) \in X \times Y$. In many scenarios, the only way to ensure perfect accuracy is to launch a costly inspection procedure on every (x, y), whereas performing the procedure $|X| \cdot |Y|$ times is prohibitively expensive. It is, therefore, important to design an algorithm that carries out the procedure on only some pairs, and renders the verdicts on the other pairs automatically with as few mistakes as possible. This article describes an algorithm that achieves the purpose using the methodology of *active monotone classification.* The algorithm ensures an asymptotically optimal tradeoff between the number of pairs inspected and the number of mistakes made.

1. INTRODUCTION

Given two sets of entities X and Y, *entity matching* aims to decide whether x and y form a *match*, i.e., whether they represent the same entity, for each pair $(x, y) \in X \times Y$. For example, X (or Y) can be a set of advertisements placed at *Amazon* (or *eBay*, resp.), where each advertisement has attributes like `prod-name`, `prod-description`, `year`, `price`, and so on. The goal is to decide whether advertisements x and y are about the same product, for all $(x, y) \in X \times Y$.

What makes the problem challenging is that the aforementioned decision cannot be made through a simple comparison on the attributes of x and y, because even a pair of matching x and y may still disagree on the attribute values. This is obvious for `prod-description` and `price` since x and y can introduce the same product in different ways, and price it differently. In fact, x and y may not agree even on a "supposedly standardized" attribute like `prod-name` (e.g., x.`prod-name` = "MS Word" vs. y.`prod-name` = "Microsoft Word Processor"), although it would be reasonable to expect x.`year` = y.`year` because advertisements are required to be correct.

In applications like the above one, the only way to ensure perfect accuracy is to call upon human experts to inspect each pair $(x, y) \in X \times Y$. This is, however, expensive because the amount of manual efforts, e.g., reading advertisements x and y in detail, is huge. It is, therefore, important to design an algorithm that asks humans to look at only some pairs, and renders the verdicts on the other pairs automatically, perhaps at the expense of a small number of errors.

Towards the above purpose, a dominant methodology behind existing approaches (e.g., [1, 3, 6, 7, 12, 14, 21, 22, 24, 25]) is to transform the task into a multidimensional classification problem with the following preprocessing:

1. First, shrink the set of all possible pairs to a subset $T \subseteq X \times Y$, by eliminating the pairs that obviously cannot be matches. This is known as *blocking*, which is carried out based on application-dependent heuristics. This step is optional; if skipped, then $T = X \times Y$. In the Amazon-eBay example, T may involve only those advertisement pairs (x, y) with x.`year` = y.`year`.

2. For each remaining entity pair $(x, y) \in T$, create a multidimensional point $p_{(x,y)}$ using a number d of similarity functions

$$sim_1, sim_2, \ldots, sim_d,$$

each evaluated on a certain feature. The i-th coordinate of $p_{(x,y)}$ equals $sim_i(x, y)$: a higher value means that x and y are more similar on the i-th feature. This creates a d-dimensional point set $P = \{p_{(x,y)} \mid (x, y) \in T\}$.

In our example, from a numerical attribute such as `price`, one may extract a feature that equals $-|x.\texttt{price} - y.\texttt{price}|$ (the purpose of the negation is to be consistent with "larger means more similar"). From a text attribute (such as `prod-name` and `prod-description`) one may extract a feature by evaluating the similarity between the corresponding texts of x and y using an appropriate metric, e.g., edit distance for short texts, and cosine similarity for long texts. Multiple feature dimensions may be derived even on the same attribute. For instance, one can extract two features by computing the edit-distance and Jaccard-distance of x.`prod-name` and y.`prod-name` separately.

Every point $p_{(x,y)} \in P$ carries a *label*, which is 1 if (x, y) is

The original version of this article was published in PODS 2018, titled "Entity Matching with Active Monotone Classification".

[*]This work was partially supported by a direct grant (Project Number: 4055079) from CUHK and by a Faculty Research Award from Google.

a match, or 0 otherwise. The original entity matching task on X and Y is thus converted to inferring the labels of the points in P. Still, human inspection is the last resort for revealing the label of each $p_{(x,y)}$ with no errors.

1.1 Problem Formalization

Let P be a set of points in $\mathbb{R}^d$, where $\mathbb{R}$ is the real domain, and d is a positive integer. Each point $p \in P$ carries a label, denoted as $label(p)$, which equals either 0 or 1. The point labels need not follow any geometric patterns, namely, $label(p)$ can be 0 or 1 regardless of the labels of the other points.

All the labels are hidden at the beginning. There is an *oracle* which an algorithm can call to disclose the label of a point $p \in P$ selected by the algorithm. When this happens, we say that p is *probed*. The algorithm's *cost* is defined as the number of points probed.

A *classifier* is a function $\mathcal{F} : \mathbb{R}^d \to \{0, 1\}$. Its *error* on P is the number of points mis-labeled, namely:

$$error(\mathcal{F}, P) = |\{p \in P \mid \mathcal{F}(p) \neq label(p)\}|.$$

$\mathcal{F}$ is *monotone* if $\mathcal{F}(p) \geq \mathcal{F}(q)$ for any points $p, q \in P$ satisfying $p[i] \geq q[i]$ on all $i \in [1, d]$, where $p[i]$ is the i-th coordinate of p. Denote by $\mathbb{C}_{mono}$ the set of all possible monotone classifiers; note that $|\mathbb{C}_{mono}|$ is infinite.

We consider two problems closely related to *active learning*:

Problem 1 (Active Monotone Classification): Find a monotone classifier $\mathcal{F}$ with small $error(\mathcal{F}, P)$ by paying a low probing cost.

Problem 2 (Active Monotone Classification with Exemptions): Probe a small set Z of points in P to find a monotone classifier $\mathcal{F}$ with small $error(\mathcal{F}, P \setminus Z)$.

Note that the two problems differ in whether the set Z of points probed is exempted from calculating the error of the returned classifier $\mathcal{F}$. The challenges behind these problems can be seen from the following extreme solutions:

- For Problem 1, one can simply probe all the points of P, paying the worst possible cost $|P|$, and then take all the time needed to find the best classifier $\mathcal{F}^* \in \mathbb{C}_{mono}$ with the smallest error on P (we do not explicitly constrain CPU computation). Note that the error of $\mathcal{F}^*$ need *not* be zero because P may not—actually most likely will not—fully obey monotonicity. In the other extreme, we can choose to probe nothing and return some classifier $\mathcal{F}$ by "wild guessing", which has the smallest cost 0, but risks suffering from the worst error $|P|$ for $\mathcal{F}$. The main challenge is to achieve the lowest error with as few probes as possible.

- For Problem 2, we can trivially achieve the minimum value 0 for $error(\mathcal{F}, P \setminus Z)$ by probing all the points of P, noticing that in this case $P \setminus Z = \emptyset$. In the other extreme, one can return a wild guess $\mathcal{F}$ with no probes at all, but again may suffer from the largest error $|P|$ for $error(\mathcal{F}, P \setminus Z)$. The main challenge is to strike an attractive tradeoff between $|Z|$ and $error(\mathcal{F}, P \setminus Z)$.

The above definitions extend in a natural way to a randomized algorithm A_{ran}. Both the classifier $\mathcal{F}$ returned by A_{ran} and the set Z of points probed by A_{ran} are random variables. For Problem 1, the *expected error* of A_{ran} is defined as $\mathbf{E}[error(\mathcal{F}, P)]$, and its *expected cost* as $\mathbf{E}[|Z|]$. Likewise, for Problem 2, the *expected error* of A_{ran} is defined as $\mathbf{E}[error(\mathcal{F}, P \setminus Z)]$, and its *expected cost* still as $\mathbf{E}[|Z|]$. In all cases, expectation is over the random choices made by the algorithm.

Remarks. The input P to both problems corresponds to the set of points obtained from the set T in the entity matching framework explained earlier. Problems 1 and 2 are designed for two different scenarios that arise frequently in practice:

- Scenario 1: the entity sets X and Y are "training sets" that represent the distributions D_X and D_Y of entities to be encountered from two sources, respectively. The purpose of finding a classifier $\mathcal{F}$ is to apply it on new $(\tilde{x}, \tilde{y}) \notin X \times Y$ to be received online where $\tilde{x}$ (or $\tilde{y}$) follows D_X (or D_Y, resp.). That $\mathcal{F}$ is accurate on P (a.k.a. T) implies that $\mathcal{F}$ should also work statistically well on $(\tilde{x}, \tilde{y})$. This is the application backdrop of Problem 1.

- Scenario 2: unlike the previous scenario, here X and Y are already the "ground sets". In other words, there are no future pairs, such as $(\tilde{x}, \tilde{y})$ in Scenario 1, to be cared for; and it suffices to match only the elements of X and Y. Thus, the "overall accuracy" of $\mathcal{F}$ on all the points in P is unimportant because if a point already has its label revealed, it does not need to be guessed by $\mathcal{F}$, and thus should be excluded from error calculation. This is the application backdrop behind Problem 2.

The rationale behind monotonicity is that, if x and y form a match according to the features picked, then any pair (x', y') more similar than (x, y) on *every* feature should also be regarded as a match. Indeed, any classifier that defies monotonicity is *awkward* because it indicates that at least some of the features have been selected inappropriately.

1.2 Relevant Research

This subsection will first give an introduction to several key findings in active learning that are relevant to Problems 1 and 2, and then review the existing entity matching solutions we are aware of.

Active Learning. Classification is a fundamental topic in machine learning. Let U be a possibly infinite set of points in $\mathbb{R}^d$. The input is an infinite stream of pairs $(p, label(p))$ where p is a point from U, and $label(p)$ is its binary label, i.e., either 0 or 1. Each pair is sampled independently from an unknown distribution D on $U \times \{0, 1\}$. A *classifier* is a function $\mathcal{F} : U \to \{0, 1\}$, whose *error probability* equals

$$\mathbf{Pr}\{p \sim D \mid \mathcal{F}(p) \neq label(p)\}$$

namely, the probability of wrongly predicting the label of a point p drawn from D. Let $\mathbb{C}$ be a *candidate class* of classifiers, and ν be the smallest error probability of all the classifiers in $\mathbb{C}$. The learning objective is the achieve the following *probabilistically approximately correct* (PAC) guarantee:

With probability at least $1 - \delta$, find a classifier $\mathcal{F}$ from $\mathbb{C}$ with error probability at most $\nu + \epsilon$, where $\delta > 0$ and $\epsilon > 0$ are input parameters.

In the more traditional *passive* setup, $label(p)$ is directly disclosed in every pair $(p, label(p))$, where the efficiency goal is to minimize the *sample cost*, which equals the number of stream pairs that an algorithm needs to see before ensuring the PAC guarantee. In practice, deciding the label of a point can be so expensive that its cost far dominates the cost of learning. This motivated *active* learning, where point labels are all hidden originally. Given an incoming point p, an algorithm can choose whether to *probe* p, i.e., paying a unit cost for the revelation of $label(p)$. The primary efficiency goal is to perform the least number of probes; efforts should still be made to avoid a high sample cost, although this now becomes a secondary goal.

Active learning has been extensively studied; see excellent surveys [16, 23]. A main challenge is to identify the *intrinsic parameters* that determine the *label complexity*, i.e., the number of probes mandatory to ensure the PAC guarantee. Considerable progress has been made in various scenarios [4, 8, 15]. Our subsequent discussion will concentrate on *agnostic* active learning, where (i) $\nu > 0$, meaning that even the best classifier in $\mathbb{C}$ cannot perfectly separate points of the two labels because, intuitively, D has "noise", and (ii) no assumptions are made on that noise. This is the branch of active learning most relevant to our work.

The state-of-the-art understanding on agnostic learning is based on two intrinsic parameters:

- the *VC dimension* λ of $\mathbb{C}$ on U;
- the *disagreement coefficient* θ of $\mathbb{C}$ under D.

We will not delve into the precise definitions of the above parameters (the interested reader may see [24] for details). For this article, it suffices to understand that a higher λ or θ indicates the necessity of probing more labels.

The dominant solution to agnostic active learning is an algorithm named A^2. Its initial ideas were developed by Balcan et al. [2], and have been substantially improved/extended subsequently [4, 9, 16]. As shown in [16], the algorithm achieves the PAC guarantee with a probing cost of

$$\tilde{O}\left(\theta \cdot \lambda \cdot \frac{\nu^2}{\epsilon^2}\right) \tag{1}$$

where $\tilde{O}(.)$ hides factors polylogarithmic to $\theta, 1/\epsilon$, and $1/\delta$. On the lower bound side, extending an earlier result of Kaariainen [17], Beygelzimer et al. [4] proved that the probing cost needs to be

$$\Omega\left(\frac{\nu^2}{\epsilon^2} \cdot \left(\lambda + \log\frac{1}{\delta}\right)\right). \tag{2}$$

There is a gap of θ between the upper and lower bounds. When this parameter is $O(1)$, the two bounds match up to polylog factors. Indeed, most success stories in the literature are based on candidate classes $\mathbb{C}$ and distributions D that give rise to a small θ (e.g., see [8, 10, 13, 26]).

Unfortunately, as will be explained later in Section 1.3, the class $\mathbb{C}_{mono}$ of monotone classifiers can have a very high VC dimension λ, and simultaneously, a very large disagreement coefficient θ. The consequences are two-fold:

- The θ gap between (1) and (2) becomes significant, suggesting that agnostic active learning has not been well understood on monotone classifiers.
- With both θ and λ being large, the A^2 algorithm incurs expensive probing costs on $\mathbb{C}_{mono}$, and may no longer be attractive.

The reader may have noticed that Problem 1 can be cast as agnostic active learning by setting U (in active learning) to P (in Problem 1), and generating an input stream (for active learning) by repeatedly sampling P. This makes A^2 a viable solution to entity matching. We will discuss its performance guarantees in relation to our results in the next subsection.

Entity Matching. There is a rich literature on entity matching; see [1, 3, 5–7, 12, 14, 18–22, 25] and the references therein. Most of these works focused on designing heuristics that perform well in practice, instead of establishing theoretical bounds. The papers [1, 3] are exceptions. In [1], Arasu et al. observed that entity matching can be approached using active learning. They presented algorithms to solve some subproblems that arose in their framework. Unfortunately, their overall solutions do not have attractive bounds for Problem 1 or 2. In [3], Bellare et al. showed that if one can solve Problem 1, the algorithm can be utilized to attain small errors of other types, e.g., those based on recalls and precisions, under certain assumptions.

1.3 Our Contributions

An Intrinsic Parameter. Recall that Problems 1 and 2 are defined on a set P of points in $\mathbb{R}^d$. Given two points $p, q \in P$, we say that p *dominates* q if $p[i] \geq q[i]$ holds on all $i \in [1, d]$. Notice that a point dominates itself by this definition. The dominance relation

$$R = \{(p, q) \in P \times P \mid p \text{ dominates } q\}$$

is a poset (partially ordered set).

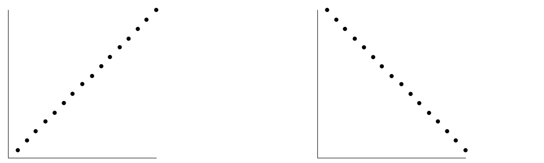
It turns out that an intrinsic parameter characterizing the hardness of both problems is the *width* w of R. Formally, w is the size of the largest $S \subseteq P$ such that no two different points in S dominate each other; we will sometimes refer to it as the *dominance width* of P. Any one-dimensional P has $w = 1$. When $d \geq 2$, w can be anywhere between 1 and n; see Figure 1 for two extreme examples.

Problem 1. Denote by $\mathcal{F}^*$ an optimal monotone classifier on P, namely, this is a classifier in $\mathbb{C}_{mono}$ with the smallest error on P. Set $k = error(\mathcal{F}^*, P)$ and $n = |P|$ throughout the article. Our first main result is:

THEOREM 1. *For Problem 1:*

- *there is a randomized algorithm that has expected error at most $2k$, and probes $O(w(1 + \log\frac{n}{w}))$ points in expectation;*
- *there exists a constant c such that, when $w \geq 2$ and $k \leq cn/w$, any algorithm with expected error $O(k)$ must have an expected probing cost of $\Omega(w \log\frac{n}{(k+1)w})$.*

Several observations can be made. First, when $k = 0$—the *noise-free* scenario where the label-1 points of P can be perfectly separated from the label-0 ones by a monotone classifier—our algorithm in the first bullet always returns such a classifier. Second, when $w = \Omega(n)$, our lower bound in the second bullet evaluates to $\Omega(n)$, meaning that the



(a) A point set of width 1 (b) A point set of width n

Figure 1: Illustration of dominance width

naive solution of probing all points in P can no longer be improved by more than a constant factor in cost, for the purpose of ensuring the smallest error asymptotically. Third, we improve the aforementioned naive solution as long as $w = o(n)$, because for such w the upper bound in the first bullet is $o(n)$. Fourth, our upper and lower bounds nearly match each other for $k = O(n/w)$. Remember that no algorithms can have an expected error less than k. Therefore, under $k = O(n/w)$, our solution is asymptotically optimal in both expected error and expected probing cost.

As explained in Section 1.2, one can apply the A^2 algorithm of agnostic active learning to solve Problem 1 by repeatedly sampling from P uniformly. To see its performance, let us fit in the appropriate values for ν, ϵ, λ, and θ, as are defined in Section 1.2. First, $\nu = k/n$, according to the definition of $error(\mathcal{F}^*, P)$. Second, to match our expected error $2k$, ϵ should be no more than $\nu = k/n$; setting ϵ to this value makes $\nu^2/\epsilon^2 = 1$. As we proved in [24], when $k = O(n/w)$, both θ and λ can reach w even in 2D space. By (1), A^2 has expected probing cost $\tilde{O}(w^2)$, which Theorem 1 improves by a factor of $\tilde{O}(w)$.

Note that (2) does *not* give a lower bound on Problem 1. Recall that (2) applies to agnostic active learning which is just one possible way to approach Problem 1.

Problem 2. If an algorithm returns a monotone classifier $\mathcal{F}$ by probing a set Z of points, it always holds that $error(\mathcal{F}, P \setminus Z) \le error(\mathcal{F}, P)$. Hence, Theorem 1 implies:

COROLLARY 1. *For Problem 2, there is a randomized algorithm that has expected error at most $2k$, and expected probing cost $O(w(1 + \log \frac{n}{w}))$.*

What is intriguing is the opposite: can we substantially reduce the error of Problem 2 without significantly increasing the probing cost? This, subtly, is a question on the usefulness of *exempting Z* from the error calculation. After all, the intended purpose of Z is to push $error(\mathcal{F}, P \setminus Z)$ *below* k (recall that, in Problem 1, the best achievable error is k). Unfortunately, we are able to show:

THEOREM 2. *Fix any integers k and n such that $k \ge 1$, and n/k is an integer at least 2. There is a set S of one-dimensional (i.e., $d = 1$) inputs to Problem 1 with the same n and k such that any algorithm guaranteeing an expected error at most $k/2$ on every input in S must entail an expected probing cost of $\Omega(n/k)$ on at least one input in S.*

Corollary 1 and Theorem 2 together point out a surprising fact. If we are satisfied with an expected error of $2k$, the expected probing cost is no more than $O(w(1 + \log \frac{n}{w}))$ universally for *all* values of k. Even better, in the context of Theorem 2 where $d = 1$, w equals 1, making $O(w(1 + \log \frac{n}{w})) = O(\log n)$. However, if we demand an expected error of $k/2$, the expected probing cost surges to $\Omega(n/k)$, which is worse than $O(\log n)$ for any $k = o(n/ \log n)$.

Theorem 2 also implies that, for $k \le n/(w \log_2 \frac{n}{w})$, the expected error must be at least $\Omega(k)$ if the expected probing cost has to be $O(w \log(n/w))$. Thus, on those values of k, our algorithm in Corollary 1 is already asymptotically optimal. Phrased differently, subject to $O(w \log \frac{n}{w})$ expected probing cost, the hardness of the problem comes from guessing the labels of points that have not been probed, such that excluding Z from error calculation makes little difference.

Content of This Article. We will focus on establishing the upper bound for Problem 1 in Theorem 1 by describing our algorithm and its analysis in full. The proofs for the lower bounds in Theorems 1 and 2, which can be found in [24], are omitted from the article.

2. THE PROPOSED ALGORITHM

Our algorithm for Problem 1—named *random probe with elimination* (RPE)—can be described in 6 lines as shown in Figure 2. If Z is the set of points probed, the algorithm produces a classifier $\mathcal{F}$ defined as:

$$\mathcal{F}(p) = \begin{cases} 1 & \text{if } p \text{ dominates a label-1 point in } Z \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

To illustrate, consider that P consists of the 16 points in Figure 3, where the black (or white) points carry label 1 (or 0, resp.). Here, k equals 3, noticing that no monotone classifiers can have an error 2 or less, while it is easy to design a monotone classifier with error 3, e.g., such a classifier would correctly capture the labels of all points except p_1, p_{11}, and p_{15}. Assume that, at Step 2, RPE happens to probe p_1 first, after which it eliminates the entire P except p_6, p_7, and p_8. Suppose also that, when Step 2 is executed again, the algorithm chooses to probe p_8, which removes all the remaining points in P. With $Z = \{p_1, p_8\}$, the classifier of (3) has an error 5 because it incorrectly maps p_2, p_3, p_5, p_{11}, and p_{15} to 1.

LEMMA 1. *The classifier $\mathcal{F}$ in (3) is monotone.*

PROOF. Suppose that there exist points p, q such that p dominates q, but $\mathcal{F}(p) = 0$ and $\mathcal{F}(q) = 1$. By (3), $\mathcal{F}(q) = 1$ means that Z has a label-1 point that is dominated by q, and hence, also dominated by p. This contradicts $\mathcal{F}(p) = 0$. □

The next lemma, together with (3), indicates a sense of symmetry between labels 0 and 1 with respect to the classifier $\mathcal{F}$ returned by RPE.

LEMMA 2. *For any $p \in P$, $\mathcal{F}(p) = 0$ if and only if p is dominated by a label-0 point in Z.*

```
Algorithm RPE($P$)
/* $P$ is the input set of Problem 1 */
1. while $P \neq \emptyset$
2.     pick a point $p$ from $P$ uniformly at random
3.     probe $p$
4.     if $label(p) = 1$ then
5.         discard from $P$ the points dominating $p$
       else
6.         discard from $P$ the points that $p$ dominates
```

Figure 2: The RPE algorithm

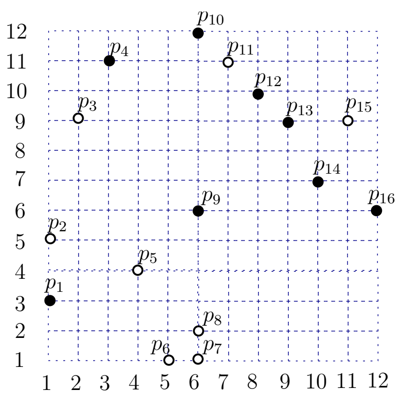


Figure 3: An input to Problem 1 where $k = 3$; black and white points have label 1 and 0, respectively.

PROOF. By (3), $\mathcal{F}(p) = 0$ means that p does not dominate any label-1 point in Z. Hence, the deletion of p from P must have been triggered by RPE probing a label-0 point p' dominating p (note that p' could be p and that a point is considered to dominate itself). This proves the "only-if" direction.

To establish the "if" direction, we first need to prove that Z obeys monotonicity, namely, for any $p, q \in Z$, if p dominates q, then $label(p) \geq label(q)$. Assume, on the contrary, that this is not true, meaning that $label(p) = 0$ and $label(q) = 1$. If p was probed before q, then q must have been discarded after discovering that p has label 0. Likewise, if q was probed before p, then p must have been discarded after discovering that q has label 1. This contradicts the fact that both p and q were probed.

The monotonicity of Z suggests that, if p is dominated by a label-0 point in Z, p cannot dominate any label-1 point in Z. Hence, $\mathcal{F}(p) = 0$, thus establishing the "if" direction. □

RPE can be implemented in $O(n \operatorname{polylog} n)$ time for fixed dimensionality d. By maintaining P in a binary search tree, we can draw a random point $p \in P$ at Step 2 in $O(\log n)$ time, such that in total we spend $O(n \log n)$ time on this step. By maintaining a range tree [11] on P, Steps 5 and 6 can be implemented in $O(x \log^d n)$ time, if x points are eliminated from P. Each point contributes to the x-term exactly once (it can be deleted only once). Hence, the total CPU time spent on these two steps is bounded by $O(n \log^d n)$.

The above is everything that a practitioner needs to know in order to apply the RPE algorithm in practice. We will delve into the theory behind RPE in the next two sections.

3. ANALYZING THE COST OF RPE

An Attrition-and-Elimination Game. We will now take a detour to discuss a relevant problem. Consider the following game between two players Alice and Bob. At the beginning, Alice is given the set $S = \{1, 2, \ldots, s\}$ for some integer $s \geq 1$. The game goes in *rounds*. In each round:

- Bob performs "attrition" first, by either doing nothing or arbitrarily deleting some elements from S.
- Alice then performs "elimination" by picking a number $p \in S$ uniformly at random, and deleting from S all the numbers larger than or equal to p.

The game ends when S becomes empty. The number of rounds is a random variable depending on Bob's strategy. How should Bob play in order to maximize the expectation of that variable?

It is fairly intuitive that Bob should do nothing at all in every round, in which case the expected number of rounds is $\Theta(1 + \log s)$. To prove this, denote by function $f(s)$ the *largest* real number such that Bob has a strategy to make the expected number of rounds equal $f(s)$. Consider the first attrition of Bob. Clearly, what matters is the number x of elements that Bob decides to remove (what those elements actually are is not relevant due to symmetry). If $x < s$, the game continues for Alice to work on a set S of size $s - x$. After her elimination, S has i elements—for each $i \in [0, s - x - 1]$—left with probability $1/(s - x)$. Therefore:

$$f(s) = 1 + \max_{x=0}^{s-1} \left\{ \frac{1}{s-x} \sum_{i=0}^{s-x-1} f(i) \right\}.$$

As the base case, $f(0) = 0$. Solving the recurrence gives $f(s) = O(1 + \log s)$ for $s \geq 1$, regardless of the value of x.

Dominance Width and Chain Decomposition. Let us return to Problem 1. Given a non-empty subset C of the input P, we say that C is:

- A *chain*, if it is possible to linearize the points of C into a sequence $p_1, p_2, \ldots, p_{|C|}$ such that p_{i+1} dominates p_i for every $i \in [1, |C| - 1]$. We will refer to the sequence as the *ascending order* of C.
- An *anti-chain*, if none of the points in C dominate each other.

In Figure 3, $\{p_6, p_8, p_{10}\}$ is a chain, whereas $\{p_4, p_{12}, p_{13}, p_{14}\}$ is an anti-chain.

A *chain decomposition* is a collection of disjoint chains C_1, C_2, ..., C_t (for some $t \geq 1$) whose union equals P. How to determine the smallest number t is a fundamental result in order theory:

> **Dilworth's Theorem:** Consider any poset; let w be the largest size of all anti-chains. Then, (i) there is a chain decomposition that contains w chains, and (ii) no chain decompositions can have less than w chains.

For instance, the input set P of Figure 3 can be divided into 6 chains: $C_1 = \{p_1, p_2, p_3, p_4, p_{10}\}$, $C_2 = \{p_{11}\}$, $C_3 = \{p_5, p_9, p_{12}\}$, $C_4 = \{p_{16}\}$, $C_5 = \{p_{13}\}$, and $C_6 = \{p_6, p_7, p_8, p_{14}, p_{15}\}$. This is a smallest chain decomposition due to the anti-chain $\{p_{10}, p_{11}, p_{12}, p_{16}, p_{13}, p_{14}\}$. Hence, the dominance width w of P is 6.

Probing Cost of RPE. Let $\{C_1, C_2, \ldots, C_w\}$ be an arbitrary smallest chain decomposition (which is *not* known to RPE). We will prove that in expectation RPE probes $O(1 + \log |C_i|)$ points in C_i for all $i \in [1, w]$. It will then follow that the total expected number of probes is $O(\sum_{i=1}^{w}(1 + \log |C_i|))$, which peaks at $O(w(1 + \log \frac{n}{w}))$ when all the chains have the same size n/w.

Without loss of generality, let us focus on C_1. Break C_1 into (i) the set C_1^{true} of points with label 1, and (ii) the set C_1^{false} of points with label 0. Due to symmetry, it suffices to prove that RPE probes $O(1 + \log |C_1^{true}|)$ points from C_1^{true} in expectation.

Set $s = |C_1^{true}|$. List the points of C_1^{true} in ascending order $p_1, p_2, \ldots, p_s$. The operations that RPE performs on this chain can be captured as an attrition-and-elimination game on an initial input $S = \{p_1, p_2, \dots, p_s\}$:

- Bob formulates his strategy by observing the execution of RPE. Suppose that the algorithm probes a point $p \notin C_1^{true}$, and shrinks P at Step 5 or 6. Bob deletes from S all those points of C_1^{true} that are discarded in the shrinking.

- When RPE probes a point $p \in C_1^{true}$, Bob finishes his attrition in this round, and passes S to Alice. *Conditioned* on $p \in C_1^{true}$, p was chosen uniformly at random from the *current* S, i.e., the set of points from C_1^{true} that are still in P. Hence, p can be regarded as the choice of Alice. When RPE shrinks P at Step 5, Alice discards p, as well as all the points behind p, from S. This finishes a round of the game. The control is passed back to Bob to start the next round.

By our earlier analysis on the attrition-and-elimination game, RPE probes $O(1 + \log s)$ points from C_1^{true} in expectation. This establishes the upper bound in Theorem 1 on the cost of RPE.

4. ANALYZING THE ERROR OF RPE

We now proceed to analyze the number of points mislabeled by the classifier $\mathcal{F}$ returned by RPE.

Fix an arbitrary optimal monotone classifier $\mathcal{F}^*$, i.e., $k = error(F^*, P)$. Henceforth, a point $p \in P$ is said to be an *ordinary* point if $\mathcal{F}^*(p) = label(p)$, or a *noise* point, otherwise. Define:

$$\begin{aligned} G_1^* &= \{p \in P \mid label(p) = 1 \text{ and } p \text{ is ordinary}\} \\ G_0^* &= \{p \in P \mid label(p) = 0 \text{ and } p \text{ is ordinary}\}. \end{aligned}$$

Since P unions G_1^*, G_0^*, and the k noise points, we know:

$$error(\mathcal{F}, P) \le error(\mathcal{F}, G_1^*) + error(\mathcal{F}, G_0^*) + k. \quad (4)$$

Let k_0 be the number of label-0 noise points, that is, points p satisfying $label(p) = 0$ but $\mathcal{F}^*(p) = 1$. The rest of the section serves as a proof of:

LEMMA 3. *$\mathbf{E}[error(\mathcal{F}, G_1^*)]$ is at most k_0.*

By the symmetry shown in Lemma 2, the above lemma implies that $\mathbf{E}[error(\mathcal{F}, G_0^*)]$ is at most the number k_1 of label-1 noise points. Equation (4) then gives $\mathbf{E}[error(\mathcal{F}, P)] \le k_0 + k_1 + k = 2k$, thus establishing the upper bound in Theorem 1 on the error of RPE.

4.1 RPE by Permutation

To analyze $error(\mathcal{F}, G_1^*)$, it will be convenient to consider an alternative implementation of RPE named *RPE-perm*, which is described in Figure 4. Compared to RPE, RPE-perm differs only in how randomization is injected: this is now done by randomly permuting P. We defer the proof of the lemma below to Section 4.4.

LEMMA 4. *RPE and RPE-perm have the same expected error and expected probing cost on every input P.*

```
Algorithm RPE-perm(P)
1.  randomly permute the points of P
    /* if a point $p \in P$ is the $i$-th ($i \in [1, n]$) in the
    permutation, define its rank $r(p)$ to be $i$ */
2.  while $P \neq \emptyset$
3.      pick the point $p \in P$ with the smallest rank
4.      probe $p$
5.      if $label(p) = 1$ then
6.          discard from $P$ the points dominating $p$
        else
7.          discard from $P$ the points that $p$ dominates
```

Figure 4: The permutation-version of RPE

4.2 Influence of Noise Points

Let us first gain some intuition on why $error(\mathcal{F}, G_1^*)$ is small in expectation. Consider the example in Figure 3, and the optimal $\mathcal{F}^*$ that mis-labels only p_1, p_{11}, and p_{15}. Here, G_1^* consists of all the black points except p_1. The bad news is that, if noise point p_{15} is probed first, the classifier $\mathcal{F}$ output by RPE will map the ordinary points p_9, p_{13}, p_{14} to 0 incorrectly (see Lemma 2), causing an increase of 3 to $error(\mathcal{F}, G_1^*)$. The good news is that, if p_{15} is probed after *any* of the ordinary points p_9, p_{13}, p_{14}, then p_{15} will be discarded and can do no harm. Under a random permutation, p_{15} has only 1/4 probability to rank before all of p_9, p_{13}, p_{14}, which seems to suggest that p_{15} could trigger an increase of only 3/4 to $error(\mathcal{F}, G_1^*)$ in expectation.

Unfortunately, the analysis is not as simple as this, due to the presence of noise point p_{11}, which complicates the conditions for p_{15} to be probed. For example, observe that, if p_{11} did not exist, p_{15} can never be probed when p_9 ranks before p_{15}. This is no longer true with the presence of p_{11}. To see this, imagine that p_{11} ranks before p_9, which in turn ranks before p_{15}. The probing of p_{11} evicts p_9 from P. On the other hand, p_{15} remains in P, and hence, gets a chance to be probed later.

The above issue arises because p_9 is dominated—and thereby is "influenced"—by both noise points p_{11} and p_{15}. *Separating* and *quantifying* the influence of each noise point turns out to be the most crucial idea behind our analysis. Let N_0 be the set of label-0 noise points, i.e., $k_0 = |N_0|$. Next, we will describe a way to calculate the "exclusive influence" $I(q)$ of each point $q \in N_0$. In particular, we will do so incrementally by observing how RPE-perm executes.

At the beginning, initialize $I(q) = 0$ for every $q \in N_0$. Whenever RPE-perm is *about* to probe a point $q \in N_0$, capture the set—denoted as $P(q)$—of points that are still in P at this moment. Then, finalize $I(q)$ as:

$$I(q) \quad = \quad \text{the number of points in } P(q) \cap G_1^* \text{ that are dominated by } q.$$

At the end of RPE-perm, if a point $q \in N_0$ is never probed, define $P(q) = \emptyset$ and finalize its $I(q)$ to be 0.

The next lemma explains why the set $\{I(q) \mid q \in N_0\}$ separates and quantifies the influence of the noise points in N_0.

LEMMA 5. $\sum_{q \in N_0} I(q) = error(\mathcal{F}, G_1^*)$.

PROOF. Consider an arbitrary $q \in N_0$ that was probed by RPE-perm. Let p be any point in $P(q) \cap G_1^*$ that is dominated by q. By Lemma 2, $\mathcal{F}(p) = 0$ because of q.

Thus, p contributes 1 to $\mathit{error}(\mathcal{F}, G_1^*)$. Hence, $\sum_{q \in N_0} I(q) \leq \mathit{error}(\mathcal{F}, G_1^*)$.

Conversely, let p be a point contributing 1 to $\mathit{error}(\mathcal{F}, G_1^*)$, that is, $\mathit{label}(p) = 1$ but $\mathcal{F}(p) = 0$. Let S be the set of label-0 points in Z that dominate p. By Lemma 2, $|S| \geq 1$. Define q to be the point in S that was probed the earliest. Because p is an ordinary point with label 1 dominated by q, q must be a noise point, i.e., $q \in N_0$. Next, we argue that p must be in $P(q)$, meaning that p contributes 1 to $I(q)$, which in turn indicates $\mathit{error}(\mathcal{F}, G_1^*) \leq \sum_{q \in N_0} I(q)$.

On the contrary, suppose that $p \notin P(q)$. Thus, p had already disappeared when RPE-perm was about to probe q. By definition of q, this implies that RPE-perm had probed a label-1 point dominated by p; but doing so should have got q discarded, giving a contradiction. □

4.3 Proof of Lemma 3

Let us denote the points of N_0 as $q_1, q_2, \ldots, q_{k_0}$ in an arbitrary order, and introduce a random variable

$$X = \sum_{i=1}^{k_0} I(q_i).$$

We will show $\mathbf{E}[X] \leq |N_0| = k_0$, which will prove Lemma 3 by way of Lemma 5. Given a subset S of P and any point $q \in P$, define:

$$D_S(q) = \{p \in S \mid \ q \text{ dominates } p\}.$$

Our proof of $\mathbf{E}[X] \leq k_0$ is inductive on k_0.

4.3.1 The Base Case

Let us start with the case $k_0 = 1$, namely, $N_0 = \{q_1\}$.

Lemma 6. *$I(q_1) > 0$ only if q_1 has a smaller rank than all the points in $D_{G_1^*}(q_1)$.*

Proof. Suppose that $D_{G_1^*}(q_1)$ has a point p that ranks before q_1 in the permutation. We argue that RPE-perm will not probe q_1.

Suppose that RPE-perm probes q_1. Consider the moment right before the probing happens. Point p must have disappeared from P (otherwise, RPE-perm cannot have chosen to probe q since the rank of p is smaller). Could it have been discarded due to the probing of a label-0 point $p' \neq q_1$? No, because otherwise, $p \in G_1^*$ asserts that p' must also be a label-0 noise point, contradicting $k_0 = 1$. Thus, p must have been discarded due to the probing of a label-1 point that p dominates. But this should have evicted q_1 as well, also giving a contradiction. □

Hence, $I(q_1) > 0$ with a probability at most $1/(1+|D_{G_1^*}(q_1)|)$. Since $I(q_1)$ obviously cannot exceed $|D_{G_1^*}(q_1)|$, we have:

$$\mathbf{E}[I(q_1)] \leq \frac{|D_{G_1^*}(q_1)|}{1 + |D_{G_1^*}(q_1)|} < 1.$$

4.3.2 The Inductive Case

Assuming $\mathbf{E}[X] \leq k_0$ when $k_0 = t - 1$ for some integer $t \geq 2$, we will prove that the same holds also for $k_0 = t$.

Define $J(i)$ ($i \in [1, t]$) as the event that q_i has the largest permutation rank among $q_1, q_2, \ldots, q_t$. We will show

$$\mathbf{E}[X \mid J(i)] \quad \leq \quad t \qquad (5)$$

for all i, which will give

$$\mathbf{E}[X] \quad = \quad \sum_{i=1}^{t} \mathbf{E}[X \mid J(i)] \cdot \mathbf{Pr}[J(i)] \leq \sum_{i=1}^{t} t \cdot \frac{1}{t} = t$$

as is needed to complete the inductive argument.

Due to symmetry, the subsequent discussion will prove (5) only for $i = t$, and hence, will be conditioned on the event $J(t)$. Recall that RPE-perm probes points in ascending order of rank. Let us define the *watershed moment* as:

- The moment right before RPE-perm probes the *first* point with a *larger* rank than all of $q_1, q_2, \ldots, q_{t-1}$;
- End of RPE-perm, if it does not probe any point that ranks after $q_1, q_2, \ldots, q_{t-1}$.

At the watershed moment, $I(q_1), \ldots, I(q_{t-1})$ have been finalized. Set $Y = \sum_{i=1}^{t-1} I(q_i)$. Denote by P_{water} the content of P at this instant.

The inductive assumption implies that $\mathbf{E}[Y] \leq t - 1$. To understand why, imagine deleting q_t from P, after which the input set P' has $t - 1$ label-0 noise points, but the same G_1^*. The permutation after removing q_t is a random permutation of P'. Thus, Y is exactly the value of $\mathit{error}(\mathcal{F}, G_1^*)$ on P'.

The remainder of the proof shows $\mathbf{E}[I(q_t) \mid J(t)] \leq 1$. This will establish (5) because

$$\mathbf{E}[X \mid J(t)] = \mathbf{E}[Y] + \mathbf{E}[I(q_t) \mid J(t)].$$

$I(q_t) = 0$ when q_t is not in P_{water} (and hence, will not be probed). Hence, it suffices to prove

$$\mathbf{E}[I(q_t) \mid J(t), q_t \in P_{water}] \leq 1.$$

Towards the purpose, we expand the left hand side over all possible sets W that P_{water} may be equal to:

$$\begin{aligned} & \mathbf{E}[I(q_t) \mid J(t), q_t \in P_{water}] \\ = \ & \sum_{W} \mathbf{E}[I(q_t) \mid J(t), q_t \in P_{water} = W] \cdot \mathbf{Pr}[W]. \qquad (6) \end{aligned}$$

We will concentrate on proving that

$$\mathbf{E}[I(q_t) \mid J(t), q_t \in P_{water} = W] \leq 1$$

regardless of W, with which (6) can be bounded from above by $\sum_W \mathbf{Pr}[W] = 1$.

Subject to the joint event "$J(t)$ and $q_t \in P_{water} = W$", the elements of W are symmetric with respect to their *relative* ordering in the permutation: any of the $|W|!$ orderings can take place with an equal probability. The analysis of $\mathbf{E}[I(q_t)]$ under that joint event is essentially the same as the base case. By the same argument as in the proof of Lemma 6, we assert that $I(q_t) > 0$ only if q_t ranks before all the points in $D_{W \cap G_1^*}(q_t)$, which happens with a probability of $1/(1 + |D_{W \cap G_1^*}(q_t)|)$. As $I(q_t)$ cannot exceed $|D_{W \cap G_1^*}(q_t)|$ under the joint event, we conclude that $\mathbf{E}[I(q_t) \mid J(t), q_t \in P_{water} = W]$ is no more than

$$\frac{|D_{W \cap G_1^*}(q_t)|}{1 + |D_{W \cap G_1^*}(q_t)|} \leq 1.$$

4.4 Proof of Lemma 4

Both RPE and RPE-perm can be described as a randomized decision tree T defined as follows. Each node u of T is associated with a subset of P, denoted as $u(P)$. If u is the root, $u(P) = P$, whereas if u is a leaf, $u(P) = \emptyset$. An internal node u has $|u(P)|$ child nodes. Each directed edge (u, v) from u to a child v stores a point—denoted as $point(u, v)$—of $u(P)$. Every point of $u(P)$ is stored on one and exactly one outgoing edge of u. For each child v, the set $v(P)$ is determined as:

- If $label(p) = 0$, $v(P)$ is the set of points in $u(P)$ that are not dominated by $point(u, v)$ (recall that a point dominates itself).
- If $label(p) = 1$, $v(P)$ is the set of points in $u(P)$ that do not dominate $point(u, v)$.

Each root-to-leaf path π represents a possible probing sequence of RPE or RPE-perm. Specifically, for each node u on π, $u(P)$ represents the content of P after the algorithm probes the points stored on (the edges of) the root-to-u path. We will prove that, for every leaf z of T, RPE and RPE-perm reach z with exactly the same probability. This establishes Lemma 4 because both error and probing cost are determined by the sequence of points probed.

Let $u_1, u_2, \ldots, u_\ell$ be the nodes on the root-to-z path (u_1 is the root and $z = u_\ell$). Obviously, RPE reaches z with probability $\Pi_{i=1}^{\ell-1} \frac{1}{|u_i(P)|}$. It remains to show that this is also true for RPE-perm.

The execution of RPE-perm is a function of the permutation of P—denoted as P_{perm}—obtained at Step 1. For each node u of T, denote by $S(u)$ the set of all possible P_{perm} that will bring the execution to u. When u is the root, $S(u)$ is the set of all $n!$ permutations.

Lemma 7. *For $i \in [2, \ell]$, $S(u_i)$ is the set of permutations $\pi \in S(u_{i-1})$ such that $point(u_{i-1}, u_i)$ has the smallest rank in π among all the points in $u_{i-1}(P)$.*

Proof. We prove the claim by induction. It holds for $i = 2$ because RPE-perm descends from u_1 (the root) to u_2 only when $point(u_1, u_2)$ is the first point of P_{perm}. Inductively, assume that the claim is true for $i = j - 1$. As mentioned before, $u_{j-1}(P)$ is the content of P after RPE-perm probes the points stored on the root-to-u_{j-1} path. Hence, the algorithm branches to u_j only if $point(u_{j-1}, u_j)$ is the next to pick in P_{perm} among the points in $u_{j-1}(P)$. So the claim holds also for $i = j$. $\square$

The lemma indicates that $|S(u_i)| = |S(u_{i-1})|/|u_{i-1}(P)|$. Hence, $|S(u_\ell)| = |S(u_1)| \cdot \Pi_{i=1}^{\ell-1} \frac{1}{|u_i(P)|}$. The probability that RPE-perm reaches u_ℓ equals $|S(u_\ell)|/n!$ which is simply $\Pi_{i=1}^{\ell-1} \frac{1}{|u_i(P)|}$. This concludes the proof of Lemma 4.

5. REFERENCES

[1] A. Arasu, M. Götz, and R. Kaushik. On active learning of record matching packages. In *SIGMOD*, pages 783–794, 2010.

[2] M. Balcan, A. Beygelzimer, and J. Langford. Agnostic active learning. *JCSS*, 75(1):78–89, 2009.

[3] K. Bellare, S. Iyengar, A. G. Parameswaran, and V. Rastogi. Active sampling for entity matching with guarantees. *TKDD*, 7(3):12:1–12:24, 2013.

[4] A. Beygelzimer, S. Dasgupta, and J. Langford. Importance weighted active learning. In *ICML*, pages 49–56, 2009.

[5] G. D. Bianco, R. Galante, M. A. Goncalves, S. D. Canuto, and C. A. Heuser. A practical and effective sampling selection strategy for large scale deduplication. *TKDE*, 27(9):2305–2319, 2015.

[6] P. Christen, D. Vatsalan, and Q. Wang. Efficient entity resolution with adaptive and interactive training data selection. In *ICDM*, pages 727–732, 2015.

[7] X. Chu, I. F. Ilyas, and P. Koutris. Distributed data deduplication. *PVLDB*, 9(11):864–875, 2016.

[8] S. Dasgupta. Coarse sample complexity bounds for active learning. In *NIPS*, pages 235–242, 2005.

[9] S. Dasgupta, D. J. Hsu, and C. Monteleoni. A general agnostic active learning algorithm. In *NIPS*, pages 353–360, 2007.

[10] S. Dasgupta, A. T. Kalai, and C. Monteleoni. Analysis of perceptron-based active learning. *Journal of Machine Learning Research (JMLR)*, 10:281–299, 2009.

[11] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008.

[12] V. Efthymiou, G. Papadakis, G. Papastefanatos, K. Stefanidis, and T. Palpanas. Parallel meta-blocking for scaling entity resolution over big heterogeneous data. *Information Systems*, 65:137–157, 2017.

[13] Y. Freund, H. S. Seung, E. Shamir, and N. Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, 28(2-3):133–168, 1997.

[14] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. W. Shavlik, and X. Zhu. Corleone: hands-off crowdsourcing for entity matching. In *SIGMOD*, pages 601–612, 2014.

[15] S. Hanneke. A bound on the label complexity of agnostic active learning. In *ICML*, pages 353–360, 2007.

[16] S. Hanneke. Theory of disagreement-based active learning. *Foundations and Trends in Machine Learning*, 7(2-3):131–309, 2014.

[17] M. Kaariainen. Active learning in the non-realizable case. In *Proceedings of International Conference on Algorithmic Learning Theory (ALT)*, pages 63–77, 2006.

[18] L. Kolb, A. Thor, and E. Rahm. Load balancing for mapreduce-based entity resolution. In *ICDE*, pages 618–629, 2012.

[19] P. Konda, S. Das, P. Suganthan, A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. F. Naughton, S. Prasad, G. Krishnan, R. Deep, and V. Raghavendra. Magellan: Toward building entity matching management systems. *PVLDB*, 9(12):1197–1208, 2016.

[20] H. Kopcke and E. Rahm. Frameworks for entity matching: A comparison. *DKE*, 69(2):197–210, 2010.

[21] H. Kopcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. *PVLDB*, 3(1):484–493, 2010.

[22] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *SIGKDD*, pages 269–278, 2002.

[23] B. Settles. Active learning literature survey. *Technical Report, University of Wisconsin-Madison*, 2010.

[24] Y. Tao. Entity matching with active monotone classification. In *PODS*, pages 49–62, 2018.

[25] A. Thor and E. Rahm. MOMA - A mapping-based object matching system. In *CIDR*, pages 247–258, 2007.

[26] L. Wang. Smoothness, disagreement coefficient, and the label complexity of agnostic active learning. *Journal of Machine Learning Research*, 12:2269–2292, 2011.

Technical Perspective: Efficient Query Processing for Dynamically Changing Datasets

Wim Martens
University of Bayreuth, Germany

The paper *Efficient Query Processing for Dynamically Changing Datasets*, by Muhammad Idris, Martín Ugarte, Stijn Vansummeren, Hannes Voigt, and Wolfgang Lehner studies two central aspects of answering queries: (1) enumerating the answers to a query and (2) changing data. It is based on two papers by the same authors or a subset thereof, namely *The Dynamic Yannakakis Algorithm: Compact and Efficient Query Processing Under Updates* [4] and *Conjunctive Queries with Inequalities Under Updates* [5].

In a nutshell, these papers show how theoretical ideas for enumerating the answers to a query (e.g., in [1]) can be brought to the point where they actually work in practice and, furthermore, can deal with updates to the data. The fact that this was possible was not at all clear from the original theoretical work, which makes the current work extremely valuable to our community. Idris et al. show in their experiments that their algorithms perform consistently better than competitor incremental view maintenance systems with up to two orders of magnitude improvements in both time and memory consumption. They also show how the algorithms can be extended to deal with more general join conditions. So, they don't just work, they actually work *really well.*

In a classic paper from 2007, Bagan et al. studied for which conjunctive queries it is possible to enumerate the answers in *constant delay*, after *linear precomputation*. This means that an algorithm is first allowed to spend linear time in the database for computing a data structure, from which it is then possible to generate the answers of the query such that the time interval between consecutive answers does not depend on the size of the data. The entire complexity analysis is done in *data complexity*, which means that it only takes the size D of the database into account and considers the size Q of the query to be a constant. This means, more concretely, that a run-time of $2^{O(Q)} \cdot O(D)$ would be considered to be linear in the database — this fact may clarify to the attentive reader why bringing such an approach to practice may indeed be challenging. A main contribution of Bagan et al. is the result that says that, if acyclic queries are *free-connex*, then they can be evaluated with linear precomputation and constant delay. However, if they are not (and fulfill mild additional constraints [1]), then they cannot be evaluated within this time bound (under the conjecture that Boolean $n \times n$ matrices cannot be multiplied in time $O(n^2)$).

Idris et al.'s work provides an algorithm that computes a data structure that depends on the data and the query, and supports constant-delay enumeration for answering the query. The algorithm does not only support (projection-free) join queries, but also free-connex acyclic conjunctive queries which is, by results of Bagan et al. [1] and Brault-Baron [3], the largest class of conjunctive queries for which such an algorithm is possible under complexity-theoretical assumptions. The approach is heavily based on Yannakakis' algorithm for acyclic conjunctive query evaluation [6].

The approach is also robust under updates in the sense that tuple insertions or deletions can be propagated efficiently into the data structure. For so-called *q-hierarchical* queries, it is able to deliver (1) constant-delay enumeration of query results and (2) update propagation in time linear in the size of the update. Berkholz et al. [2] proved that the q-hierarchical queries are precisely the conjunctive queries that allow such an algorithm, unless the Online Matrix-Vector Multiplication conjecture is false.

This is a paper written by researchers with solid backgrounds in systems and theory *and it shows.* It provides algorithms for query evaluation that match rather tightly with theoretical lower bounds and perform very well in the experimental settings.

1. REFERENCES

[1] G. Bagan, A. Durand, and E. Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *Computer Science Logic (CSL)*, pages 208–222, 2007.

[2] C. Berkholz, J. Keppeler, and N. Schweikardt. Answering conjunctive queries under updates. In *Symposium on Principles of Database Systems (PODS)*, pages 303–318, 2017.

[3] J. Brault-Baron. *De la pertinence de l'énumération: complexité en logiques propositionelle et du premier ordre*. PhD thesis, Université de Caen, 2013.

[4] M. Idris, M. Ugarte, and S. Vansummeren. The dynamic Yannakakis algorithm: Compact and efficient query processing under updates. In *International Conference on Management of Data (SIGMOD)*, pages 1259–1274, 2017.

[5] M. Idris, M. Ugarte, S. Vansummeren, H. Voigt, and W. Lehner. Conjunctive queries with inequalities under updates. *PVLDB*, 11(7):733–745, 2018.

[6] M. Yannakakis. Algorithms for acyclic database schemes. In *International Conference on Very Large Data Bases*, pages 82–94, 1981.

Efficient Query Processing for Dynamically Changing Datasets [*]

Muhammad Idris
Université Libre de Bruxelles &
TU Dresden
muhammad.idris@ulb.ac.be

Martín Ugarte
PUC Chile
martinugarte@uc.cl

Stijn Vansummeren
Université Libre de Bruxelles
svsummer@ulb.ac.be

Hannes Voigt
Neo4J
hannes.voigt@neo4j.com

Wolfgang Lehner
TU Dresden
wolfgang.lehner@tu-dresden.de

ABSTRACT

The ability to efficiently analyze changing data is a key requirement of many real-time analytics applications. Traditional approaches to this problem were developed around the notion of Incremental View Maintenance (IVM), and are based either on the materialization of subresults (to avoid their recomputation) or on the recomputation of subresults (to avoid the space overhead of materialization). Both techniques are suboptimal: instead of materializing results and subresults, one may also maintain a data structure that supports efficient maintenance under updates and from which the full query result can quickly be enumerated. In two previous articles, we have presented algorithms for dynamically evaluating queries that are easy to implement, efficient, and can be naturally extended to evaluate queries from a wide range of application domains. In this paper, we discuss our algorithm and its complexity, explaining the main components behind its efficiency. Finally, we show experiments that compare our algorithm to a state-of-the-art (Higher-order) IVM engine, as well as to a prominent complex event recognition engine. Our approach outperforms the competitor systems by up to two orders of magnitude in processing time, and one order in memory consumption.

1 Introduction

The ability to efficiently analyze changing data is a key requirement of many real-time analytics applications like Stream Processing [20], Complex Event Recognition [9], Business Intelligence [17], and Machine Learning [22].

In this context, we tackle the problem of *dynamic query evaluation*, where a given query Q has to be evaluated against a database that is constantly changing. Concretely, when database db is updated to database $db + u$ under update u, the objective is to efficiently compute $Q(db + u)$, taking into consideration that $Q(db)$ was already evaluated and re-computations could be avoided.

Dynamic query evaluation is of utmost importance if response time requirements for queries under concurrent data updates have to be met or if data volumes are so large that full re-evaluation of queries based on raw data is prohibitive.

The following example illustrates our setting. Assume that we wish to detect potential credit card fraud. Credit card transactions specify their timestamp (ts), account number (acc), and amount ($amnt$). A typical fraud pattern is that, in a short period of time, a criminal tests a stolen credit card with a few small purchases to then make larger purchases (cf. [18]). Assuming that the short period of time is 1 hour, this pattern could be detected by dynamically evaluating the query in Figure 1. Queries like this may exhibit arbitrary local predicates and multi-way joins with equality as well as inequality predicates. Traditional techniques to process such queries dynamically can be categorized in two approaches: *relational* and *automaton-based*. We outline the core principles of relational approaches in the following and refer to [13] for an in-depth discussion of the drawbacks of automaton-based approaches.

Relational approaches such as [2, 10, 16] build upon the technique of *Incremental View Maintenance (IVM)* [7]. To process a query Q over a database db, IVM techniques materialize the output $Q(db)$ and evaluate *delta queries* [10]. Upon update u, delta queries use db, u, and the materialized $Q(db)$ to compute the set of tuples to add/delete from $Q(db)$ in order to obtain $Q(db + u)$. If u is small with respect to the database db, this is expected to be faster than recomputing $Q(db + u)$ from scratch. To further speed up dynamic query processing, also the result of some subqueries of Q may be redundantly materialized. This approach is known as Higher-Order IVM (HIVM) [15, 16].

Unfortunately, (H)IVM shows a serious drawback in terms of additional memory overhead, which quickly becomes prohibitive for interactive data analytics scenarios: materialization of $Q(db)$ requires $\Omega(|Q(db)|)$ space, where $|db|$ denotes the size of db. Therefore, when $Q(db)$ is large, which is often the case in data preparation scenarios for training statistical models, materializing $Q(db)$ quickly becomes impractical, especially for main-memory based systems. HIVM is even more affected by this problem than IVM since it not only

[*]This paper surveys the results of two papers: "The Dynamic Yannakakis Algorithm: Compact and Efficient Query Processing Under Updates" [12], which was published in SIGMOD Conference 2017 and "Conjunctive Queries with Inequalities Under Updates" [13], published in PVLDB 11(7), 2018. During the work described, Martín Ugarte was affiliated to ULB and Hannes Voight to TU Dresden.

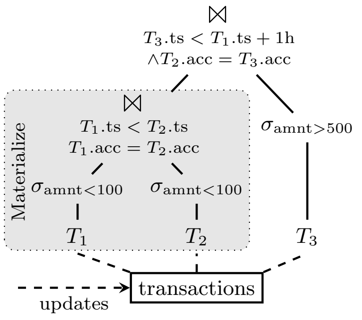


Figure 1: Example query for detecting fraudulent credit card activity.

materializes the result of Q but also the results of some subqueries. In fact some of these subresults can be partial join results, which can be larger than both db and $Q(db)$. For example, in our fraud query, HIVM would materialize the results of the join in the shaded area in Figure 1. Intuitively, this join builds the table of all pairs of *small* transactions that could be part of a credit card fraud if a third relevant transaction occurs. Therefore, if we assume that there are N small transactions in the time window, all of the same account, this materialization will take $\Theta(N^2)$ space. This naturally becomes impractical when N grows.

While these problems are inherent to (H)IVM methods, they can be avoided by taking a different approach to dynamic query evaluation: instead of materializing $Q(db)$ we can build a succinct data structure that (1) supports efficient maintenance under updates and (2) *represents* $Q(db)$ in the sense that from it we can *generate* $Q(db)$ as efficiently as if it were materialized. In particular, the representation is equipped with index structures so that we can enumerate $Q(db)$ *with constant delay* [19]: one tuple at a time, while spending only a constant amount of work to produce each new tuple. This makes the enumeration competitive with enumeration from materialized query results.

In essence, we hence separate dynamic query processing into two stages: (1) an update stage where we only maintain under updates the (small) information that is necessary for result enumeration and (2) an enumeration stage where the query result is efficiently enumerated.

In our work, which is documented in detail in [12] and [13], we are concerned with designing a practical family of algorithms for dynamic query evaluation based on this idea for queries featuring both equi-joins and inequality joins, as well as certain forms of aggregation. Our main insight is that, for acyclic conjunctive queries, such algorithms can naturally be obtained by modifying Yannakakis' seminal algorithm for processing acyclic joins in the static setting [23].

In a first step, we address the problem of efficiently evaluating acyclic aggregate-join queries by providing the *Dynamic Yannakakis Algorithm* (DYN) [12]. The representation of query results that underlies this algorithm has several desirable properties:

- (P_1) It allows to enumerate $Q(db)$ with constant delay.
- (P_2) For any tuple $\vec{t}$, it can be used to check whether $\vec{t} \in Q(db)$ in constant time.
- (P_3) It requires only $\mathcal{O}(|db|)$ space and is hence independent of the size of $Q(db)$.
- (P_4) it features efficient maintenance under updates: given update u to db, we can update the representation of $Q(db)$ to a representation of $Q(db+u)$ in time $\mathcal{O}(|db|+|u|)$. In contrast, (H)IVM may require $\Omega(|u|+|Q(db+u)|)$ time in the worst case. For the subclass of q-hierarchical queries [4], our update time is $\mathcal{O}(|u|)$.

Based on this technique to dynamically process queries with equi-joins, we provide the core intuiton of a generalization of the Dynamic Yannakakis Algorithm to conjunctive queries with arbitrary θ-joins. We show that, in the specific case of inequality joins, this generalization improves the state of the art for dynamically processing inequality joins by performing consistently better, with up to two orders of magnitude improvements in processing time and one order in memory consumption.

It is important to note that we consider query evaluation in main memory and measure time and space under data complexity [21]. That is, the query is considered to be fixed and not part of the input. This makes sense under dynamic query evaluation, where the query is known in advance and the data is constantly changing.

2 Preliminaries

Query Language. Throughout the paper, let $x, y, z, \ldots$ denote *variables* (also commonly called *column names* or *attributes*). A *hyperedge* is a finite set of variables. We use $\overline{x}, \overline{y}, \ldots$ to denote hyperedges. A *Generalized Conjunctive Query* (GCQ) is an expression of the form

$$Q = \pi_{\overline{y}}\left(r_1(\overline{x_1}) \bowtie \cdots \bowtie r_n(\overline{x_n}) \mid \bigwedge_{i=1}^{m} \theta_i(\overline{z_i})\right).$$

Here $r_1, \ldots, r_n$ are *relation symbols*; $\overline{x_1}, \ldots, \overline{x_n}$ are hyperedges (of the same arity as $r_1, \ldots, r_n$); $\theta_1, \ldots, \theta_m$ are predicates over $\overline{z_1}, \ldots, \overline{z_m}$, respectively; and both $\overline{y}$ and $\bigcup_{i=1}^{m} \overline{z_i}$ are subsets of $\bigcup_{i=1}^{n} \overline{x_i}$. We treat predicates abstractly: for our purpose, a predicate over $\overline{x}$ is a (not necessarily finite) decidable set θ of tuples over $\overline{x}$. For example, $\theta(x, y) = x < y$ is the set of all tuples (a, b) satisfying $a < b$. We indicate that θ is a predicate over $\overline{x}$ by writing $\theta(\overline{x})$. Throughout the paper, we consider only non-nullary predicates with $\overline{x} \neq \emptyset$.

Example 2.1. The following query is a GCQ.

$$\pi_{y,z,w,u}\left(r(x, y) \bowtie s(y, z, w) \bowtie t(u, v) \mid x < z \wedge w < u\right)$$

Intuitively, the query asks to take the natural join of $r(x, y)$ and $s(y, z, w)$, form the cartesian product with $t(u, v)$, and subsequently select those tuples that satisfy $x < z$ and $w < u$.

We call $\overline{y}$ the *output variables* of Q and denote them *out*(Q). If $\overline{y} = \overline{x_1} \cup \cdots \cup \overline{x_n}$ then Q is called *full* and we may omit the symbol $\pi_{\overline{y}}$ for brevity. We denote by *full*(Q) the full *GCQ* obtained from Q by setting *out*(Q) to $\overline{x_1} \cup \cdots \cup \overline{x_n}$. The elements $r_i(\overline{x_i})$ are called *atoms*. *at*(Q) denotes the set of all atoms in Q, and *pred*(Q) the set of all predicates in Q. A *conjunctive query* (or CQ) is a GCQ where *pred*(Q) $= \emptyset$.

Semantics. We evaluate GCQs over Generalized Multiset Relations (GMRs for short) [12, 15, 16]. Let $\mathrm{dom}(x)$ denote the domain of variable x. As usual, a tuple over $\overline{x}$ is a mapping $\vec{t}$ that assigns a value from $\mathrm{dom}(x)$ to every $x \in \overline{x}$. A GMR R over $\overline{x}$ is a function $R\colon \mathbb{T}[\overline{x}] \to \mathbb{Z}$ mapping tuples over $\overline{x}$ to integers such that $R(\vec{t}) \neq 0$ for finitely many tuples $\vec{t}$. Here, $\mathbb{T}[\overline{x}]$ denotes the set of all tuples over x. In contrast to classical multisets, the multiplicity of a tuple in a GMR can

R

x	y	z	$\mathbb{Z}$
1	2	2	2
2	4	6	3
1	2	3	3

S

u	v	$\mathbb{Z}$
4	5	5
2	3	4
1	4	2

T

u	v	$\mathbb{Z}$
4	5	−4
2	1	6
1	4	3

$S \bowtie T$

u	v	$\mathbb{Z}$
4	5	−20
1	4	6

$\pi_y(R)$

y	$\mathbb{Z}$
2	5
4	3

$S + T$

u	v	$\mathbb{Z}$
4	5	1
2	3	4
1	4	5
2	1	6

$S - T$

u	v	$\mathbb{Z}$
4	5	9
2	3	4
1	4	−1
2	1	−6

$\sigma_{y<u}(R \bowtie S)$

x	y	z	u	v	$\mathbb{Z}$
1	2	2	4	5	10
1	2	3	4	5	15

Figure 2: Operations on GMRs.

hence be negative, allowing to treat insertions and deletions uniformly. We write $var(R)$ for $\overline{x}$; $\text{supp}(R)$ for the finite set of all tuples with non-zero multiplicity in R; $\vec{t} \in R$ to indicate $\vec{t} \in \text{supp}(R)$; and $|R|$ for $|\text{supp}(R)|$. A GMR R is *positive* if $R(\vec{t}) > 0$ for all $\vec{t} \in \text{supp}(R)$. The operations of GMR union $(R + S)$, minus $(R - S)$, projection $(\pi_{\overline{z}} R)$, natural join $(R \bowtie T)$ and selection $(\sigma_P(R))$ are defined similarly as in relational algebra with multiset semantics. Figure 2 illustrates these operations; see [12, 16] for formal semantics.

A *database* over a set $\mathcal{A}$ of atoms is a function db that maps every atom $r(\overline{x}) \in \mathcal{A}$ to a positive GMR $db_{r(\overline{x})}$ over $\overline{x}$. We write $|db|$ for $\sum_{r(\overline{x}) \in \mathcal{A}} |db_{r(\overline{x})}|$. Given a database db over the atoms occurring in query Q, the evaluation of Q over db, denoted $Q(db)$, is the GMR over $\overline{y}$ constructed in the expected way: take the natural join of all GMRs in the database, do a selection over the result w.r.t. each predicate, and finally project on $\overline{y}$. It is instructive to note that after evaluation, each result tuple has an associated multiplicity that counts the number of derivations for the tuple. In other words, the query language has built-in support for `COUNT` aggregations. We note that, in their full generality, GMRs can carry multiplicities that are taken from an arbitrary algebraic semiring structure (cf., [15]), which can be useful to describe the computation of more advanced aggregations over the result of a GCQ [1]. To keep the notation and discussion simple we fix the ring $\mathbb{Z}$ of integers throughout the paper, but our results generalize to arbitrary semirings and their associated aggregations.

Updates and deltas. An *update* to a GMR R is simply a GMR ΔR over the same variables as R. Applying update ΔR to R yields the GMR $R + \Delta R$. An *update to a database* db is a collection u of (not necessarily positive) GMRs, one GMR $u_{r(\overline{x})}$ for every atom $r(\overline{x})$ of db, such that $db_{r(\overline{x})} + u_{r(\overline{x})}$ is positive. We write $db + u$ for the database obtained by applying u to each atom of db.

Computational Model. We focus on dynamic query evaluation in main-memory. We assume a model of computation where the space used by tuple values and integers, the time of arithmetic operations on integers, and the time of memory lookups are all $\mathcal{O}(1)$. We further assume that every GMR R can be represented by a data structure that allows (1) enumeration of R with constant delay (as defined in Section 3); (2) multiplicity lookups $R(\vec{t})$ in $\mathcal{O}(1)$ time given $\vec{t}$; (3) single-tuple insertions and deletions in $\mathcal{O}(1)$ time; while (4) having size that is proportional to $|R|$. We further assume the existence of dynamic data structures that can be used to index GMRs on a subset of their variables. Concretely if R is a GMR over $\overline{x}$ and I is an index of R on $\overline{y} \subseteq \overline{x}$ then we assume that for every $\overline{y}$-tuple $\vec{s}$ we can retrieve in $\mathcal{O}(1)$ time a pointer to the GMR $R \ltimes \vec{s}$, which is the GMR over $\overline{x}$ consisting of all tuples that project to $\vec{s}$:

$$R \ltimes \vec{s} \in \mathbb{GMR}[\overline{x}] \colon \vec{t} \mapsto \begin{cases} R(\vec{t}) & \text{if } \vec{t}[\overline{y}] = \vec{s} \\ 0 & \text{otherwise} \end{cases}$$

Moreover, we assume that single-tuple insertions and deletions to R can be reflected in the index in $\mathcal{O}(1)$ time and that an index takes space linear in $|R|$. Essentially, our assumptions amount to perfect hashing of linear size [8]. Although this does not directly match a realistic setting, it is well known that complexity results for this model can be translated, through amortized analysis, to average complexity in real-life implementations [8].

3 Dynamic Yannakakis

In this section we formulate Dyn, a dynamic version of the Yannakakis algorithm [23], that focuses on the evaluation of *CQs*. How to deal with θ-joins is discussed in Section 4.

3.1 Intuition

A data structure D supports *enumeration* of a set E if there is a routine ENUM such that ENUM(D) outputs each element of E exactly once. Such enumeration occurs with delay d if the time until the first output; the time between any two consecutive outputs; and the time between the last output and the termination of ENUM(D), are all bounded by d. D supports enumeration of a GMR R if it supports enumeration of the set $E_R = \{(\vec{t}, R(\vec{t})) \mid \vec{t} \in \text{supp}(R)\}$. When evaluating a GCQ Q over a database db, we will be interested in representing the elements of $Q(db)$ by means of a data structure D_{db}, such that we can enumerate $Q(db)$ from D_{db}. If, for every db, the delay to enumerate $Q(db)$ from D_{db} is independent of $|db|$ then we say that the enumeration occurs *with constant delay* [19].

As a trivial example of constant delay enumeration (CDE for short) of a GMR R, assume that the pairs $(\vec{t}, R(\vec{t}))$ of E_R are stored in an array A (without duplicates). Then A supports CDE of R: ENUM(A) simply iterates over each element in A, one by one, always outputting the current element. Since array indexation is a $\mathcal{O}(1)$ operation, this gives constant delay. This example shows that CDE of $Q(db)$ can always be done naively by materializing $Q(db)$ in an in-memory array. Unfortunately, this requires memory proportional to $|Q(db)|$ which, depending on Q, can be of size polynomial in $|db|$. We hence desire other data structures to represent $Q(db)$ using less space, while still allowing CDE.

To understand how this can be done, it is instructive to consider a simple binary join $Q = R(x, y) \bowtie S(y, z)$ and analyze why traditional join processing algorithms do not yield CDE. Suppose that we evaluate Q using a simple in-memory hash join with R as build relation and S as probe relation. Assume that the corresponding index of R on y (i.e. the hash table) has already been computed. Now observe that, when iterating over S to probe the index, we may have to visit an unbounded number of S-tuples that do not join with any R-tuple. Consequently, the delay between consecutive output tuples may be as large as $|S|$, which is not constant. A similar analysis shows that other join algorithms, such as the sort-merge join, do not yield enumeration with constant delay.

How can we obtain CDE for $R(x,y) \bowtie S(y,z)$? Intuitively speaking, if we can ensure to only iterate over those S-tuples that have matching R-tuples, we trivially obtain constant delay since then every probe will yield a new output tuple. As such, the key is to first compute $Y = \pi_y(R) \bowtie \pi_y(S)$ and index both R and S on y. We can then iterate over the elements of Y, probing both R and S in each iteration to generate the output with constant delay. In the presence of updates, this means that we only need to maintain Y, as well as the indexes on R and S—all of which are of size linear in *db* and can be maintained efficiently.

3.2 The Algorithm

To extend the intuition of Section 3.1 from a binary join to general CQs that feature both multiway equi-joins and projections, we need to maintain for all the relations that are used as *probe* relations in a join, the set of tuples that will match the corresponding build relation(s). Of course, we also need to decide in what order we will join the relations, since this determines the auxiliary sets of tuples (like Y above) that we need to maintain. For DYN, this query plan is specified by means of a pair (T, N) called a *GJT pair.*

GJT pairs. To simplify notation, we denote the set of all variables (resp. atoms, resp. predicates) that occur in an object X (such as a query) by $var(X)$ (resp. $at(X)$, resp. $pred(X)$). In particular, if X is itself a set of variables, then $var(X) = X$. We extend this notion uniformly to labeled trees. E.g., if n is a node in tree T, then $var_T(n)$ denotes the set of variables occurring in the label of n, and similarly for edges and trees themselves. If T is clear from the context, we omit subscripts from our notation.

Definition 3.1. A *GJT pair* is a tuple (T, N) with T a *generalized join tree* and N a *sibling-closed connex subset* of T. A generalized join tree (GJT) is a node-labeled directed tree $T = (V, E)$ such that:
- T is binary: every node has at most two children.
- Every leaf is labeled by an atom.
- Every interior node n is labeled by a hyperedge and has at least one child c such that $var(n) \subseteq var(c)$. Such a child is called a *guard* of n.
- Whenever the same variable x occurs in the label of two nodes m and n of T, then x occurs in the label of each node on the unique path linking m and n.

A *simple GJT* is a GJT where $var(n) \subseteq var(c)$ for every node n with child c, i.e., a GJT where every child is a guard of its parent. A *connex subset* of T is a set $N \subseteq V$ that includes the root of T such that the subgraph of T induced by N is a tree. N is *sibling-closed* if for every node $n \in N$ with a sibling m in T, m is also in N. The *frontier* of a connex set N is the subset $F \subseteq N$ consisting of those nodes in N that are leaves in the subtree of T induced by N.

Figure 3 shows a GJT pair (T_1, N_1) and a GJT T_2. T_1 is simple, but T_2 is not since $t(x,u)$ is not a guard of $\{x,y\}$. The set $N_1 = \{\{x\}, \{x,y\}, t(x,u)\}$, highlighted in gray, is a sibling-closed connex subset of T_1, and its frontier is $\{\{x,y\}, t(x,u)\}$.

Definition 3.2. Let (T, N) be a GJT pair and assume that $\{\!|r_1(\overline{x_1}), \dots, r_n(\overline{x_n})|\!\}$ is the multiset of atoms occurring as labels in the leaves of T. Then the query associated to T is the full join $\mathcal{Q}[T] = (r_1(\overline{x_1}) \bowtie \cdots \bowtie r_n(\overline{x_n}))$ and the query associated to (T, N) is the CQ $\mathcal{Q}[T, N] = \pi_{var(N)}(\mathcal{Q}[T])$.

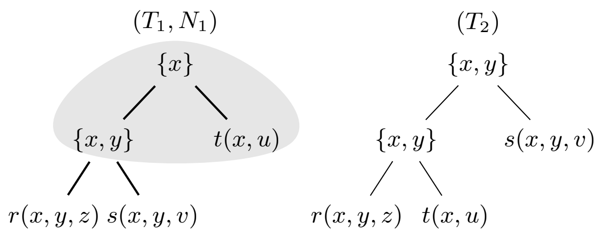
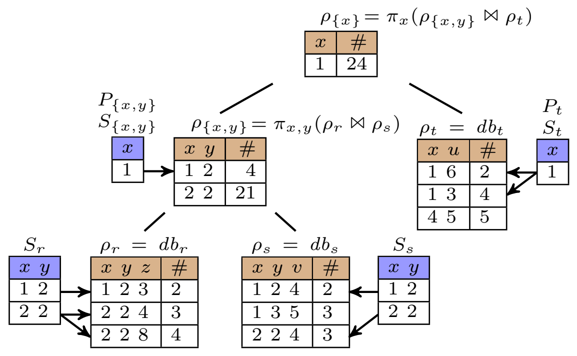


Figure 3: Two example GJTs.



$\rho_{\{x\}} = \pi_x(\rho_{\{x,y\}} \bowtie \rho_t)$
x | # ; 1 | 24
$P_{\{x,y\}}$
$S_{\{x,y\}}$
x ; 1
$\rho_{\{x,y\}} = \pi_{x,y}(\rho_r \bowtie \rho_s)$
x y | # ; 1 2 | 4 ; 2 2 | 21
$\rho_t = db_t$
x u | # ; 1 6 | 2 ; 1 3 | 4 ; 4 5 | 5
P_t
S_t
x ; 1
S_r
x y ; 1 2 ; 2 2
$\rho_r = db_r$
x y z | # ; 1 2 3 | 2 ; 2 2 4 | 3 ; 2 2 8 | 4
$\rho_s = db_s$
x y v | # ; 1 2 4 | 2 ; 1 3 5 | 3 ; 2 2 4 | 3
S_s
x y ; 1 2 ; 2 2

Figure 4: (T_1, N_1)-representation for the database *db* specified by the GMRs depicted at the leaves.

The data structure. Following the intuition of Section 3.1, a GJT pair (T, N) acts as query plan by which DYN processes $\mathcal{Q}[T, N]$ dynamically. In particular, the GJT T specifies the data structure to be maintained and drives the processing of updates, while the connex set N drives the enumeration of query results. The data structure itself is defined next.

Definition 3.3. Let (T, N) be a GJT pair and let *db* be a database over $at(Q)$. The T*-reduct* (or *semi-join reduction*) of *db* is a collection ρ of GMRs, one GMR ρ_n for each node $n \in T$, defined inductively as follows:
- if $n = r(x)$ is an atom, then $\rho_n = db_{r(x)}$
- if n has a single child c, then $\rho_n = \pi_{var(n)}\rho_c$
- otherwise, n has two children c_1 and c_2. In this case we have $\rho_n = \pi_{var(n)}(\rho_{c_1} \bowtie \rho_{c_2})$. Note that, because n has a guard child, this is actually a semijoin.

A T-reduct needs to be augmented by suitable index structures to be used for both enumeration and maintenance under updates. Concretely for each node n with parent p in T, the following indexes are created:
- If n belongs to N, then we store an index P_n of ρ_n on $var(p) \cap var(n)$, called the *parent index* of n.
- If n is a node with a sibling m, then we store an index S_n of ρ_n on $var(n) \cap var(m)$, called the *sibling index* of n.

The T-reduct ρ together with the collection of indexes is called a (T, N)*-representation* for *db*, or (T, N)-rep for short.

Figure 4 depicts an example (T_1, N_1)-representation ρ for the database *db* composed of the GMRs shown at the leaves of the tree. It is important to observe that the size of this representation for a database *db* can be at most linear in the size of *db*. The reason is that each interior node only does projections or semijoins. Therefore, as illustrated in Figure 4, for each node n there is some descendant atom

Algorithm 1 Dyn: Dynamic Yannakakis

```
1: function $\textsc{enum}_{T,N}(\rho)$
2:   for each $\vec{t} \in \rho_{\mathrm{root}(T)}$ do $\textsc{enum}_{T,N}(root(T), \vec{t}, \rho)$

3: function $\textsc{enum}_{T,N}(n, \vec{t}, \rho)$
4:   if $n$ is in the frontier of $N$ then yield $(\vec{t}, \rho_n(\vec{t}))$
5:   else if $n$ has one child $c$ then
6:     for each $\vec{s} \in \rho_c \ltimes \vec{t}$ do $\textsc{enum}_{T,N}(c, \vec{s}, \rho)$
7:   else $n$ has two children $c_1$ and $c_2$
8:     for each $\vec{t_1} \in \rho_{c_1} \ltimes \vec{t}$ do
9:       for each $\vec{t_2} \in \rho_{c_2} \ltimes \vec{t}$ do
10:        for each $(\vec{s_1}, \mu) \in \textsc{enum}_{T,N}(c_1, \vec{t_1}, \rho)$ do
11:          for each $(\vec{s_2}, \nu) \in \textsc{enum}_{T,N}(c_2, \vec{t_2}, \rho)$ do
12:            yield $(\vec{s_1} \cup \vec{s_2},\ \mu \times \nu)$

13: procedure $\textsc{update}_{T,N}(\rho, u)$
14:   for each $n \in$ leafs$(T)$ labeled by $r(\overline{x})$ do
15:     $\Delta_n \leftarrow u_{r(\overline{x})}$
16:   for each $n \in$ nodes$(T) \setminus$ leafs$(T)$ do
17:     $\Delta_n \leftarrow$ empty GMR over $var(n)$
18:   for each $n \in$ nodes$(T)$, traversed bottom-up do
19:     $\rho_n + = \Delta_n$
20:     if $n$ has a parent $p$ and a sibling $m$ then
21:       $\Delta_p + = \pi_{var(p)}\,(\rho_m \bowtie \Delta_n)$
22:     else if $n$ has parent $p$ then
23:       $\Delta_p + = \pi_{var(p)}\,\Delta_n$
```

α (possibly n itself) such that $\mathrm{supp}(\rho_n) \subseteq \mathrm{supp}(\pi_{var(n)}\, db_\alpha)$. Consequently, the indexes are also of size linear in db.

Given these definitions, the enumeration and maintenance algorithms that form the Dynamic Yannakakis algorithm are shown in Algorithm 1. They operate as follows.

Enumeration. To enumerate from a (T, N)-rep we iterate over the reductions ρ_n with $n \in N$ in a nested fashion, starting at the root and proceeding top-down. When n is the root, we iterate over all tuples in ρ_n. For every such tuple $\vec{t}$, we iterate only over the tuples in the children c of n that are compatible with $\vec{t}$ (i.e., tuples in ρ_c that join with $\vec{t}$). Note that such tuples can be enumerated efficiently thanks to the index P_c. This procedure continues until we reach nodes in the frontier of N at which time the output tuple can be constructed. The pseudocode is given by the routine ENUM in Algorithm 1, where the tuples that are compatible with $\vec{t}$ are computed by $\rho_c \ltimes \vec{t}$.

Update processing. To maintain a (T, N)-rep under update u it suffices to traverse the nodes of T in a bottom-up fashion. At each node n we have to compute the update Δ_n to apply to ρ_n and its associated indexes. For leaf nodes, this update is given by the update u itself. For interior nodes, Δ_n can be computed from the update and the original reduct of its children. Algorithm 1 gives the pseudocode. Here, line 21 is then implemented by means of a straightforward hash-join (using the sibling index S_m on ρ_m). As a side effect of modifying ρ the associated indexes are also updated (not shown).

Theorem 3.4. *Let (T, N) be a fixed GJT pair. Given a (T, N)-rep of db with T-reduct ρ, $\textsc{enum}_{T,N}(\rho)$ enumerates $\mathcal{Q}[T, N](db)$ with constant delay. Moreover, $\textsc{update}(\rho, u)$ updates the (T, N)-rep from a (T, N)-rep of db to a (T, N)-rep of $db + u$ in time $\mathcal{O}(|db| + |u|)$. If T is simple, then the update time is $\mathcal{O}(|u|)$, hence independent of $|db|$.*

Note that $\textsc{update}_{T,N}$ can be used to build a (T, N)-rep of db in time $\mathcal{O}(|db|)$: start from an empty (T, N)-rep (which represents the empty database) and then call $\textsc{update}_{T,N}(\rho, u)$ with $u = db$. This hence shows that Dyn can be used to enumerate $\mathcal{Q}[T, N](db)$ with constant delay after linear time preprocessing.

We also note that if $\vec{t}$ is a tuple over $var(M)$ for some connex subset $M \subseteq N$ of T, then checking whether $\vec{t} \in \pi_{var(M)}\mathcal{Q}[T, N](db)$ can be done in constant time: it suffices to check that $\vec{t}[var(m)] \in \rho_m$ for every $m \in M$ and return true if and only if this is the case. Since T and N are fixed, the size of M is bounded and these are a constant number of checks, all of which run in constant time.

Discussion. Dyn heavily relies on having a GJT pair (T, N) to process queries. If, for a CQ Q there exists some GJT pair (T, N) such that $Q \equiv \mathcal{Q}[T, N]$ then Q is said to be *free-connex acyclic*, and (T, N) is called a *GJT pair for Q*. Q is *acyclic* if $full(Q) \equiv \mathcal{Q}[T]$ for some T.[1] Not all CQs are (free-connex) acyclic. For instance, the triangle query $r(x, y) \bowtie s(y, z) \bowtie t(x, z)$ is the prototypical example of a non-acyclic query. Furthermore, $\pi_{x,z}(r(x, y) \bowtie s(y, z))$ is acyclic but not free-connex acyclic. Since every free-connex acyclic CQ is acyclic, this example shows that free-connex acyclic CQs form a strict subclass of the acyclic CQs. Recent analysis of query logs show that free-connex acyclic queries occur very frequently in practice [5]. We refer readers interested in algorithms for computing GJT pairs for GCQs to [14].

One may wonder whether algorithms with the same properties as Dyn can be obtained for CQs that are not free-connex acyclic. It is known that this is not possible for the class of all acyclic CQs, unless multiplication of two $n \times n$ binary matrices can be computed in $\mathcal{O}(n^2)$ time [3]. Using further complexity-theoretic assumptions, it is possible to show that this is also not possible for the class of all CQs [6].

It is also known [4] that, unless the Online Matrix-Vector Multiplication conjecture [11] is false, the class of queries that allow both (1) constant-delay enumeration of query results and (2) update processing time $\mathcal{O}(|u|)$ for every update u, is exactly the class of so-called *q-hierarchical queries*. While we forego a formal definition of this class, we show in [12] that a CQ Q is q-hierarchical if, and only if there exists a GJT pair (T, N) for Q such that T is simple. Since Dyn has update time $\mathcal{O}(|u|)$ for exactly these queries, Dyn hence meets the theoretical lower bound.

For readers familiar with the Yannakakis algorithm [23] it may not be obvious from the description above why Dyn can be claimed to be a dynamic version of Yannakakis. We refer to [12] for a discussion.

4 Dealing with θ-joins

To extend Dyn to also process θ-joins, it is instructive to consider the GCQ $Q = (R(x, y) \bowtie S(y, z) \mid x < z)$ where the θ-join is an inequality-join. To obtain CDE for Q, assume that we have already computed $Y = \pi_{x,y}(\sigma_{x<z}(R(x, y) \bowtie S(y, z)))$ and that, moreover, we have a more powerful index structure I that allows, for any tuple $\{x, y\}$-tuple $\vec{t}$ over, to enumerate $\sigma_{x<z}(S(y, z) \ltimes \vec{t})$ with constant delay. We can then obviously enumerate Q with constant delay by iterating

[1]There exists many equivalent definitions of when a join query is acyclic, and consequently also of when a CQ with projections is free-connex acyclic. See [12, 14] for a discussion of why the new definition that we give here is equivalent to the existing ones.

over the elements of $\vec{t} \in Y$, and for each such $\vec{t}$, probe I to produce the tuples $\vec{s} \in \sigma_{x<z}(S(y,z) \ltimes \vec{t})$, outputting each $\vec{s} \cup \vec{t}$. Since $\sigma_{x<z}(S(y,z) \ltimes \vec{t})$ allows CDE, the entire procedure is CDE. The key question then, is how we can build this more powerful index structure I. The solution is to build a normal (hash-based) index J of S on y but use sorting to store the index in such a way that for every $\vec{t}$ over y this index returns a pointer to $S \ltimes \vec{t}$ for which tuples are enumerated in descending order on z. This now supports CDE of $\sigma_{x<z}(S(y,z) \ltimes \vec{t})$, for every $\vec{t}$ over $\{x, y\}$: use J to enumerate $R \ltimes \vec{t}$ with constant delay and in decreasing order on z. Yield the current tuple $\vec{s}$ that is being enumerated in this fashion, provided that $\vec{t}(x) < \vec{s}(z)$. As soon as $\vec{t}(x) \geq \vec{s}(z)$ we know that all subsequent $\vec{s}$ will fail the inequality, and we can hence terminate. This example forms the basic intuition in how we can extend DYN to deal with *GCQs* with inequality joins. We next sketch the generalization to arbitrary θ-joins, and refer to [13] for detailed exposition.

First, in the presence of θ-joins, a GJT pair is defined exactly as in Definition 3.1 except that now additionally every edge $p \rightarrow c$ from parent p to child c is labeled by a set $\mathit{pred}(p \rightarrow c)$ of predicates. It is required that every predicate $\theta(\overline{z})$ in this set satisfies $\overline{z} \subseteq \mathit{var}(p) \cup \mathit{var}(c)$. The query $\mathcal{Q}[T, N]$ associated to (T, N) then becomes $\pi_{\mathit{var}(N)}(\mathcal{Q}[T] \mid \wedge_{\theta(\overline{z}) \in \mathit{pred}(T)} \theta(\overline{z}))$. Here, $\mathit{pred}(T)$ are all the predicates occurring on edges in T.

Next, (T, N)-reps are extended to account for predicates. Concretely, the inductive definition of ρ_n in the T-reduct becomes:

- if $n = r(x)$ is an atom, then $\rho_n = db_{r(x)}$
- if n has a single child c, then $\rho_n = \pi_{\mathit{var}(n)} \sigma_{\mathit{pred}(n \rightarrow c)} \rho_c$
- otherwise, n has two children c_1 and c_2. In this case we set $\rho_n = \pi_{\mathit{var}(n)} \sigma_{\mathit{pred}(n)} (\rho_{c_1} \bowtie \rho_{c_2})$.

Here $\mathit{pred}(n)$ denotes the set of all predicates on the edges from n to its children in T. The indexes that we need to maintain are modified as follows: P_n should now allow CDE of $\sigma_{\mathit{pred}(p \rightarrow n)}(\rho_n \ltimes \vec{t})$, for every $\vec{t}$ over $\mathit{var}(p)$, where p is n's parent. S_n should allow CDE of $\sigma_{\mathit{pred}(p)}(\rho_n \ltimes \vec{t})$ for every tuple $\vec{t}$ over $\mathit{var}(m)$ where m is the sibling of n. The exact design of these indexes of course depends on the semantics of the predicates included in T; for inequality predicates we have sketched above how they work. The generalization of DYN works as long as we have these indexes.

Finally, Algorithm 1 is modified so that in Lines 6, 8 and 9 we iterate over $\sigma_{\mathit{pred}(n \rightarrow c)}(\rho_c \ltimes \vec{t})$ resp. $\sigma_{\mathit{pred}(n \rightarrow c_1)}(\rho_{c_1} \ltimes \vec{t})$ and $\sigma_{\mathit{pred}(n \rightarrow c_2)}(\rho_{c_2} \ltimes \vec{t})$. Lines 19, 21, 23 are modified to compute the Δ GMRs under the now-modified definition of (T, N)-rep. We refer to the general version of DYN with arbitrary θ-joins as GDYN, and the version where all θ-joins are inequalities as IEDYN.

Theorem 4.1. *Let (T, N) be a fixed GJT pair. Given a (T, N)-rep of db, both* GDYN *and* IEDYN *correctly enumerate $\mathcal{Q}[T, N](db)$ and update the (T, N)-rep to a rep of $db + u$ under update u. In the case that all predicates in θ-joins are inequalities,* IEDYN *has the following complexity. If there is at most one inequality on each edge in T, then* IEDYN *enumerates with constant delay and has $\mathcal{O}(M \cdot \log(M))$ update time where $M = (|db| + |u|)$. If T has some edge that contains multiple inequalities, the delay is $\mathcal{O}(\log(|db|))$ and the update time is $\mathcal{O}(M^2 \cdot \log(M))$.*[2]

[2] In [13] there was an incorrect claim: we stated that updates could be processed in time $\mathcal{O}(M \cdot log(M))$ in this last case. We then found a bug in our algorithm and we currently do not know if this bound can be achieved. See [14] for a proof that IEDYN runs in the bounds claimed here.

5 Experimental Evaluation

We have implemented (IE)DYN as a query compiler that generates executable code in the Scala programming language. The generated code instantiates a (T, N)-rep for a query Q and defines *trigger functions* that are used for maintaining the (T, N)-rep under updates.

Our implementation supports two modes of operation: *push-based* and *pull-based*. In both modes, the system maintains the (T, N)-rep under updates. In the *push-based mode* the system generates, on its output stream, the delta result $\Delta Q(db, u) := Q(db + u) - Q(db)$ after each single-tuple update u. To do so, it uses a modified version of enumeration that we call *delta enumeration*. Similarly to how ENUM enumerates $Q(db)$, delta enumeration enumerates $\Delta Q(db, u)$ with constant delay (if Q has at most one inequality per pair of atoms) resp. logarithmic delay (otherwise). To do so, it uses both (1) the (T, N)-reduct GMRs ρ_n and (2) the delta GMRs $\Delta\rho_n$ that are computed by UPDATE when processing u. In this case, however, one also needs to index the $\Delta\rho_n$ similarly to ρ_n. In the *pull-based mode*, in contrast, the system only maintains the (T, N)-rep under updates but does not generate any output stream. Nevertheless, at any time a user can call ENUM to obtain the current output.

It should be noted that our implementation also supports the processing of general acyclic GCQs that are not necessarily free-connex. This is done using the following simple strategy. Let Q be acyclic but not free-connex. First, compute a free-connex acyclic approximation Q_F of Q. Q_F can always be obtained from Q by extending the set of output variables of Q. In the worst case, we need to add all variables, and Q_F becomes the full join underlying Q. Then, use (IE)DYN to maintain a (T, N)-rep for Q_F. When operating in push-based mode, for each update u, we use the (T, N)-rep to delta-enumerate $\Delta Q_F(db, u)$ and project each resulting tuple to materialize $\Delta Q(db, u)$ in an array. Subsequently, we copy this array to the output. Note that the materialization of $\Delta Q(db, u)$ here is necessary since the delta enumeration can produce duplicate tuples after projection. When operating in pull-based mode, we materialize $Q(db)$ in an array, and use delta enumeration of Q_F to maintain the array under updates. Of course, under this strategy, we require $\Omega(|Q(db)|)$ space in the worst case, just like (H)IVM would, but we avoid the (partial) materialization of delta queries. Note the distinction between the two modes: in push-based mode $\Delta Q(db, u)$ is materialized (and discarded once the output is generated), while in pull-based mode $Q(db)$ is materialized upon requests. Finally, our implementation also supports common aggregates like `SUM` and `AVG`, see [12] for more information.

5.1 Conjunctive Queries

We evaluate a subset of queries available in the industry-standard benchmarks TPC-H and TPC-DS. In particular, we evaluate those queries involving only equijoins, whose FROM-WHERE clauses are acyclic. Queries are divided into acyclic full-join queries (called FQs) and acyclic aggregate queries. Acyclic full join queries are generated by taking the FROM clause of the corresponding queries on the benchmarks. We omit the ORDER BY and LIMIT clauses, we replaced the

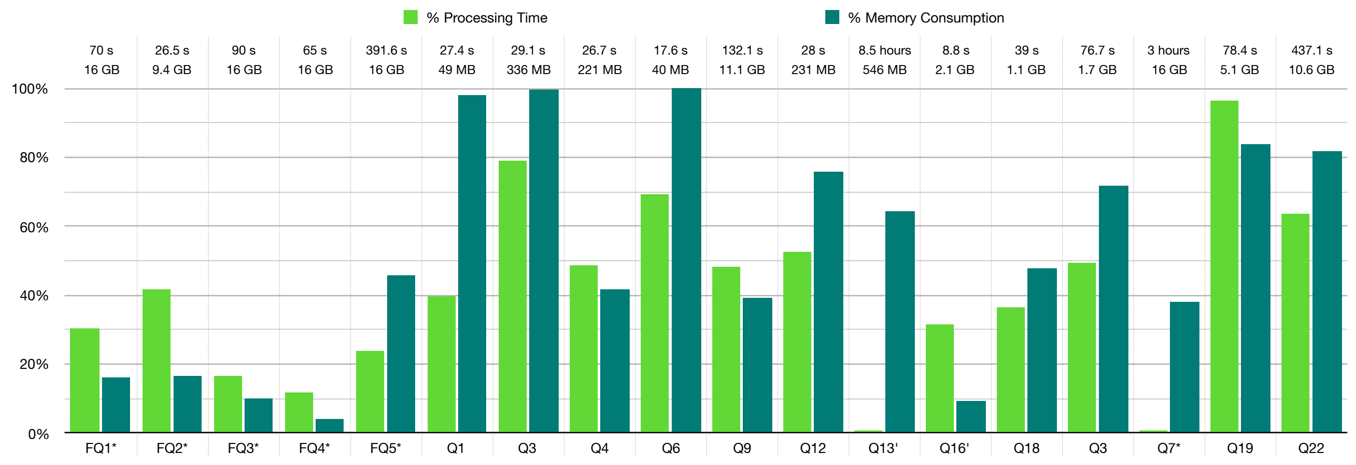


Figure 5: DYN usage of resources as a percentage of the resources consumed by DBToaster (lower is better).

Benchmark		Query	# of tuples
TPC-H	Full joins	**FQ1**	2,833,827
		FQ2	2,617,163
		FQ3	2,820,494
		FQ4	2,270,494
	Aggregate queries	**Q1**	7,999,406
		Q3	10,199,406
		Q4	9,999,406
		Q6	7,999,406
		Q9	11,346,069
		Q12	9,999,406
		Q13	2,200,000
		Q16'	1,333,330
		Q18	10,199,406
TPC-DS	Full joins	**FQ5**	10,669,570
	Aggregate queries	**Q3**	11,638,073
		Q7	13,559,239
		Q19	11,987,115
		Q22	36,138,621

Table 1: CQ benchmark stream sizes.

left-outer join in TPC-H query Q13 by an equijoin, and modified TPC-H Q16 to remove an inequality. See [12] for the full query specification.

Our workload consist of a stream of updates, where each update consists of a single-tuple insertion. The streams were generated using the TPC-H and TPC-DS data generators. The number of tuples in each stream is depicted in Table 1.

We compare IEDYN with DBToaster [16] using *memory footprint* and *update processing time* as comparison metrics. DBToaster is a state-of-the-art implementation of HIVM. It operates in pull-based mode, and is optimized for aggregations over equi-joins. DBToaster has been extensively tested for such queries and has proven to be more efficient than a commercial database management system, a commercial stream processing system and an IVM implementation [16]. It is therefore an interesting implementation to compare to. DBToaster compiles given SQL statements into executable trigger programs in different programming languages. We compare against those generated in Scala from the DBToaster Release 2.2.[3]

[3]`https://dbtoaster.github.io/`

Figure 5 depicts the resources used by DYN as a percentage of the resources used by DBToaster, both operating in pull-based mode. For each query, we plot the percentage of memory used by DYN considering that 100% is the memory used by DBToaster, and the same is done for processing time. This improves readability and normalizes the chart. To present the absolute values, on top of the bars corresponding to each query we write the memory and time used by DBToaster. Some executions of DBToaster failed because they required more than 16GB of main memory. In those cases, we report 16GB of memory and the time it took the execution to raise an exception. We mark such queries with an asterisk (*) in Figure 5. Note that DYN never runs out of memory, and times reported for DYN are the times required to process the entire update stream.

From Figure 5 we see that for full join queries (FQ1-FQ5), DYN outperforms DBToaster by close to one order of magnitude in both memory consumption and processing time, illustrating the effectiveness of maintaining (T, N)-reps rather than the query results themselves, especially when these results are large. For aggregate queries, Figure 5 shows that DYN can significantly improve the memory consumption of HIVM while improving processing time—up to two orders of magnitude for TPC-H Q13' and TPC-DS Q7. See [12] for an in-depth discussion.

While the T-reps maintained by IEDYN feature constant delay enumeration, this theoretical notion hides a constant factor that could decrease performance in practice when compared to full materialization. Experiments detailed in [12] show that this not the case: DYN's enumeration time is competitive with DBToaster.

5.2 Conjunctive Queries with Inequalities

To gauge the effectiveness of IEDYN on GCQs that feature inequality joins, we evaluate the acyclic queries listed in Table 2 on synthetically-generated streams of single-tuple insertion updates. The sizes of the update streams are intentionally kept low, since they generate huge output sizes (cf. Table 2).

Here we only compare IEDYN with Esper[4] but refer to [13] for a more detailed comparison against other state of the art

[4]`http://www.espertech.com/esper/esper-downloads/`

Query	Expression	Join	Type	|Stream|	|Output|
GCQ_1	$R(a,b,c) \bowtie S(d,e,f) \mid a < d$	$<$	Full	12k	$18,017$k
GCQ_2	$R(a,b,c) \bowtie S(d,e,f) \bowtie T(g,h,i) \mid a < d \wedge e < g$	$<$	Full	2.7k	$178,847$k
GCQ_3	$R(a,b,c) \bowtie S(d,e,f) \bowtie T(g,h,i) \mid a < d \wedge d < g$	$<$	Full	2.7k	$90,425$k
GCQ_4	$R(a,b,c) \bowtie S(d,e,f,k) \bowtie T(g,h,i,k) \mid a < d \wedge d < g$	$<,=$	Full	21k	$297,873$k
GCQ_5	$\pi_{a,b,d,e,f,g,h}(GCQ_3)$	$<$	Free-connex	2.7k	$114,561$k
GCQ_6	$\pi_{d,e,f,g,h,k}(GCQ_4)$	$<,=$	Free-connex	21k	$99,043$k
GCQ_7	$\pi_{b,c,e,f,h,i}(GCQ_3)$	$<$	Free-connex	2.7k	$114,561$k
GCQ_8	$\pi_{b,c,e,f,h,i}(GCQ_4)$	$<,=$	Free-connex	21k	$297,873$k

Table 2: GCQ benchmark queries, together with update stream and result sizes, $k = 1000$.

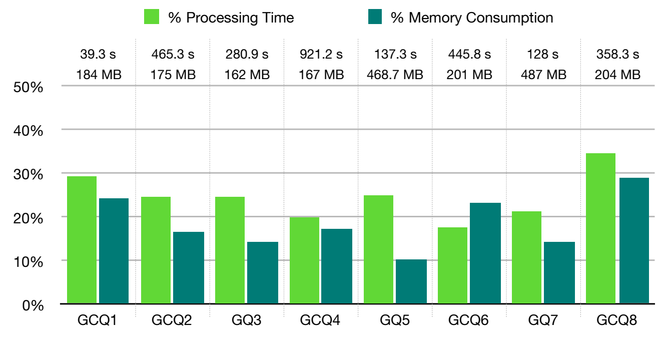


Figure 6: IEDYN resource usage as a percentage of the resources used by Esper (lower is better).

systems. Esper is a complex event processing engine with a relational model based on Stanford STREAM [2]. It operates in push-based mode. We use the Java-based open source implementation[4] for our comparisons.

Figure 6 depicts the resources used by IEDYN as a percentage of the resources used by Esper, both operating in push-based mode. IEDYN significantly outperforms Esper on all full join queries (GCQ_1–GCQ_4). We note that for these queries, even in push-based mode IEDYN can support the enumeration of query results from its data structures at any time while competing push-based systems have no such support. Hence, IEDYN is not only more efficient but also provides more functionality. IEDYN also significantly outperforms Esper on free-connex queries GCQ_5 and GCQ_6 with more than a threefold improvement in processing time and an order of magnitude improvement in memory usage on Q_7. For non-free-connex queries GCQ_7 and GCQ_8, IEDYN continues to significantly outperform Esper in processing time, showing an order of magnitude improvement in memory usage for GCQ_7.

6 Summary

Traditional techniques for dynamic query evaluation are based either on materialization (to avoid recomputation of subresults), or on recomputation of (to avoid the space overhead of materialization). We have shown that both techniques are suboptimal: instead of materializing subresults, one can use Dynamic Yannakakis to maintain a data structure that is succinct; and yet supports all operations one commonly expects from materialization: enumeration with constant delay as well as fetching single tuples in constant time. Our experiments against state-of-the art engines in different domains show that this can improve performance by orders of magnitude.

7 References

[1] M. Abo Khamis, H. Q. Ngo, and A. Rudra. FAQ: Questions asked frequently. In *Proc. of PODS*, pages 13–28, 2016.
[2] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom. STREAM: the stanford data stream management system. In *Data Stream Management - Processing High-Speed Data Streams*, pages 317–336. Springer, 2016.
[3] G. Bagan, A. Durand, and E. Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *Proc. of CSL*, pages 208–222, 2007.
[4] C. Berkholz, J. Keppeler, and N. Schweikardt. Answering conjunctive queries under updates. In *Proc. of PODS*, pages 303–318, 2017.
[5] A. Bonifati, W. Martens, and T. Timm. An analytical study of large SPARQL query logs. Full version of PVLDB 11(2) paper, under submission. Obtained through personal communication.
[6] J. Brault-Baron. *De la pertinence de l'énumération: complexité en logiques.* PhD thesis, Université de Caen, 2013.
[7] R. Chirkova and J. Yang. *Materialized Views.* Now Publishers Inc., Hanover, MA, USA, 2012.
[8] T. Cormen. *Introduction to Algorithms, 3rd Edition:*. MIT Press, 2009.
[9] G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM CSUR*, 44(3):15:1–15:62, 2012.
[10] A. Gupta, I. Mumick, and V. Subrahmanian. Maintaining views incrementally. In *Proc. of SIGMOD*, pages 157–166, 1993.
[11] M. Henzinger, S. Krinninger, D. Nanongkai, and T. Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proc. of STOC*, pages 21–30, 2015.
[12] M. Idris, M. Ugarte, and S. Vansummeren. The dynamic yannakakis algorithm: Compact and efficient query processing under updates. In *Proc. of SIGMOD*, pages 1259–1274, 2017.
[13] M. Idris, M. Ugarte, S. Vansummeren, H. Voigt, and W. Lehner. Conjunctive queries with inequalities under updates. *PVLDB*, 11(7):733–745, 2018.
[14] M. Idris, M. Ugarte, S. Vansummeren, H. Voigt, and W. Lehner. Conjunctive queries with θ-joins under updates. Technical report, 2019. Available at `http://arxiv.org/abs/1905.09848`.
[15] C. Koch. Incremental query evaluation in a ring of databases. In *Proc. of PODS*, pages 87–98, 2010.
[16] C. Koch, Y. Ahmad, O. Kennedy, M. Nikolic, A. Nötzli, D. Lupei, and A. Shaikhha. Dbtoaster: higher-order delta processing for dynamic, frequently fresh views. *VLDB Journal*, pages 253–278, 2014.
[17] B. Sahay and J. Ranjan. Real time business intelligence in supply chain analytics. *Information Management & Computer Security*, 16(1):28–48, 2008.
[18] N. P. Schultz-Møller, M. Migliavacca, and P. R. Pietzuch. Distributed complex event processing with query rewriting. In *Proc. of DEBS 2009*, 2009.
[19] L. Segoufin. Constant delay enumeration for conjunctive queries. *SIGMOD Rec.*, 44(1):10–17, 2015.
[20] M. Stonebraker, U. Çetintemel, and S. Zdonik. The 8 requirements of real-time stream processing. *SIGMOD Rec.*, 34(4):42–47, 2005.
[21] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *Proc. of STOC*, pages 137–146, 1982.
[22] W. Wang, J. Gao, M. Zhang, S. Wang, G. Chen, T. K. Ng, B. C. Ooi, J. Shao, and M. Reyad. Rafiki: Machine learning as an analytics service system. *PVLDB*, 12(2):128–140, 2018.
[23] M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. of VLDB*, pages 82–94, 1981.

Technical Perspective: Efficient Signal Reconstruction for a Broad Range of Applications

Zachary G. Ives
University of Pennsylvania
zives@cis.upenn.edu

When problems are scaled to "big data," researchers must often come up with new solutions, leveraging ideas from multiple research areas — as we frequently witness in today's big data techniques and tools for machine learning, bioinformatics, and data visualization. Beyond these heavily studied topics, there exist other classes of general problems that need to be rethought at scale. One such problem is that of *large-scale signal reconstruction* [4]: taking a set of observations of relatively low dimensionality, and using them to reconstruct a high-dimensional, unknown *signal*. This class of problems arises when we can only observe a subset of a complex environment that we are seeking to model — for instance, placing a few sensors and using their readings to reconstruct an environment's temperature, or monitoring multiple points in a network and using the readings to estimate end-to-end network traffic, or using 2D slices to reconstruct a 3D image.

This *signal reconstruction problem* (SRP) is typically approached as an optimization task, in which we search for the high-dimensional signal that minimizes a *loss function* comparing it to the known properties of the signal. Prior solutions to the SRP make use of linear algebra techniques [4] or expectation maximization [2] to find a solution. However, at scale, the dimensionality of the signal is high enough to render such optimization techniques too costly. In "Efficient Signal Reconstruction for a Broad Range of Applications," Asudeh et al. show that algorithmic insights about SRP, combined with database techniques such as similarity joins and sketches, can be used to scalably solve the signal reconstruction problem. The paper creatively integrates query processing, approximation, and linear algebra techniques.

The authors start by noting that SRP is a special case of quadratic programming, which they exploit by solving the Lagrangian dual formulation of the original problem. Building upon this, they make a connection to query processing: the key part of the algorithm computes the product of a (typically very sparse) matrix A with its transpose, AA^T. In turn, that computation derives most of its value from a small number of elements from A.

The authors creatively leverage this observation to handle huge matrices, by implementing matrix multiplication via a set-intersection primitive. They build upon set-similarity joins and apply threshold-based techniques [3] to bound the values of the matrix product, thus developing a fast approximation algorithm. Finally, they show how to use min-hash sketches [1] to approximate the sets, allowing further trade-offs of accuracy vs performance (and space). Experimental analysis shows these techniques scale well enough to to predict end-to-end routes in a large P2P network, which is several orders of magnitude larger than prior solutions could handle.

This paper is notable because it scalably addresses an under-served problem with practical impact, and does so in a clean, insightful, and systematic way. It makes several key contributions. First, it shows how insights into the linear algebra computation can be used for greater efficiency (the connection to quadratic programming, which allows it to be solved via the Lagrangian dual). Subsequently, it makes insightful connections to techniques from query processing and sketches, to develop approximation algorithms. Finally, the paper conducts an experimental study demonstrating high performance at scale. The paper illustrates the potential benefits of connecting important optimization problems with database approximate query processing techniques.

1. REFERENCES

[1] Andrei Z Broder. On the resemblance and containment of documents. In *Compression and Complexity of SEQUENCES 1997*, pages 21–29. IEEE, 1997.

[2] Jin Cao, Drew Davis, Scott Vander Wiel, and Bin Yu. Time-varying network tomography: router link data. *Journal of the American statistical association*, 95(452):1063–1075, 2000.

[3] Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*, pages 5–5. IEEE, 2006.

[4] Curtis R Vogel. *Computational methods for inverse problems*, volume 23. SIAM, 2002.

Efficient Signal Reconstruction for a Broad Range of Applications

Abolfazl Asudeh[†], Jees Augustine[‡], Azade Nazi[§], Saravanan Thirumuruganathan[¶], Nan Zhang[‖], Gautam Das[‡], Divesh Srivastava[**]
[†] University of Illinois at Chicago; [‡] University of Texas at Arlington; [§] Google Brain
[¶]QCRI, HBKU; [‖] Pennsylvania State University; [**]AT&T Labs-Research
asudeh@uic.edu, {jees.augustine@mavs, gdas@cse}.uta.edu,
azade.nazi@google.com, sthirumuruganathan@hbku.edu.qa, nan@ist.psu.edu,
divesh@research.att.com

ABSTRACT

The signal reconstruction problem (SRP) is an important optimization problem where the objective is to identify a solution to an under-determined system of linear equations $AX = b$ that is closest to a given prior. It has a substantial number of applications in diverse areas including network traffic engineering, medical image reconstruction, acoustics, astronomy and many more. Most common approaches for solving SRP do not scale to large problem sizes. In this paper, we propose a dual formulation of this problem and show how adapting database techniques developed for scalable similarity joins provides a significant speedup when the A matrix is sparse and binary. Extensive experiments on real-world and synthetic data show that our approach produces a significant speedup of up to 20x over competing approaches.

1. INTRODUCTION

The database community has been at the forefront of grappling with challenges of big data and has developed numerous techniques for the scalable processing and analysis of massive datasets. These techniques often originate from solving core data management challenges but then find their way into effectively addressing the needs of big data analytics. For example, efficiency of machine learning has been successfully boosted by database techniques as varied as materialization, join optimization, query rewriting for efficiency, query progress estimation, federated databases, etc. This paper studies how database techniques can benefit another foundational problem in big data analytics, *large-scale signal reconstruction* [22], which is of significant interest to research communities such as computer networks [25], medical imaging [11,14], etc. We demonstrate that the scalability of existing solutions can be significantly improved using ideas originally developed for similarity joins [7] and selectivity estimation for set similarity queries [3,12].

Signal Reconstruction Problem (SRP): The essence of SRP is to solve a linear system of the form $AX = b$, where X is a high-dimensional unknown *signal* (represented by an m-d vector in $\mathbb{R}^m$), b is a low-dimensional projection of X that can be observed in practice (represented by an n-d vector in $\mathbb{R}^n$ with $n \ll m$), and A is a $n \times m$ matrix that captures the linear relationship between X and b. There are many real-world applications that follow the SRP model (see § 2.1). For example, high-dimensional signals like environmental temperature can only be observed through low-dimensional observations, like readings captured by a small number of temperature sensors. Similarly, as further explained in § 2.1, end-to-end network traffic, another high-dimensional signal, is often monitored through low-dimensional readings such as traffic volume on routers in the backbone or edge networks. In these applications, the laws of physics or the topology of computer networks reveal the value of A, and our objective is to reconstruct the high-dimensional signal X from the observation b based on the knowledge of A.

Since $n \ll m$, the linear system is underdetermined. That is, for a given A and b, there are an infinite number of feasible solutions (of X) that satisfy $AX = b$ [13,22]. In order to identify the best reconstruction of the signal, it is customary to define and optimize for a *loss function* that measures the distance between the reconstructed X and a prior understanding of certain properties of X. For example, one can represent one's prior belief of X as an m-d vector X', and define the loss function as the ℓ_2-norm of $X - X'$, i.e., $\|X - X'\|_2$. In other cases, when prior knowledge indicates that X is sparse, one can define the loss function as the ℓ_0-norm of X, aiming to minimize the number of non-zero elements in the reconstructed signal. For the purpose of this paper, we consider the ℓ_2-based loss function of $\|X - X'\|_2$, which has been adopted in many application-oriented studies such as [11,25].

Running Example of SRP: While SRP has a broad range of applications, for the ease of discussion, it is important to have a running example of SRP on a domain-specific application. What we use as a running example of SRP throughout the paper is a common instance of network tomography (§ 2.1.1), where the objective is to compute the pairwise end-to-end traffic in IP Networks. Pairwise traffic measures the volume of traffic between all pairs of source-destination nodes in an IP network, and has numerous uses such as capacity planning, traffic engineering and detecting traffic anomalies. Informally, consider an IP network where various sources and destinations send different amounts of traffic to each other. The network administrator is aware of the network topology and the routing table (from which we can construct matrix A). In addition, the administrator can observe the traffic passing through each link in the backbone network (observation b). The goal is to find the amount of traffic flow between all source-destination pairs (signal X). Note that one cannot directly measure the raw traffic between all source-destination pairs due to challenges in instrumentation and storage - see [25,26] for a technical discussion. In almost all real-world IP networks, the number of source-destination pairs is significantly larger than the number of links, leading to an underdetermined linear system. To reconstruct the pairwise traffic, the network community introduced various traffic models, e.g., the

©VLDB Endowment 2018. This is a minor revision of the paper entitled "Leveraging similarity joins for signal reconstruction", published in the Proceedings of the VLDB Endowment, Vol. 11, No. 10, 1276–1288. DOI: https://doi.org/10.14778/3231751.3231752

gravity model [25], as the prior for X', and used the ℓ_2-distance between X and the prior as the loss function. Note that in reconstructing the pairwise distances, efficiency is a concern front-and-center, especially given the rise of Software Designed Networks (SDNs) which feature much larger sizes and much more frequent topological changes, pushing further the scalability requirements of signal reconstruction algorithms.

Research Gap: Because of the importance of SRP, there has been extensive work from multiple communities on finding efficient solutions. To solve the problem efficiently, methods explored in the recent literature include statistical likelihood based iterative algorithms based on expectation-maximization [5], as well as the use of linear algebraic techniques such as computing the pseudoinverse of A [22] or performing Singular Value Decomposition (SVD) on A, and iterative algorithms for solving the linear system [22]. Yet even these approaches cannot scale to fully meet the requirements in practice, especially the traffic reconstruction needs of large-scale IP networks - which call for a more scalable solution [26].

Our Approach: In this paper, we consider a special case of SRP where A, X, b are non-negative with A being a *sparse binary matrix*. Such a setting finds its applications in many domains, as explained in § 2.1.

Our proposed solution starts with an exact algorithm based on the transformation of the problem into its Lagrangian dual representation. As we shall show in § 6, our algorithm DIRECT, which directly computes X through the dual representation, already outperforms commonly used approaches for SRP, as it avoids expensive linear algebraic operations required by the previous solutions. Next, we investigate whether our approach can be sped up even further, by replacing exact computations with approximation techniques. This can be useful in applications where the user is willing to trade accuracy for efficiency. We carefully investigate the computational bottlenecks of DIRECT and find it to be a special case of matrix multiplication involving a sparse binary matrix with its transpose. We start by investigating a seemingly straightforward sampling strategy for approximately computing this matrix multiplication, but encounter a negative result. Then, we use the observation that a small number of cells in the result matrix of the bottleneck operation take the bulk of the values, and propose a threshold-based algorithm for approximating it. Specifically, we reduce the problem to computing the dot product of two vectors if and only if their similarity is above a user-provided threshold. Our key idea here is to leverage various database techniques to speed up the multiplication operation. We propose a hybrid algorithm based on a number of techniques originally proposed for computing similarity joins and selectivity estimation of set similarity queries, resulting in significant speedup in solving SRP in comparison with the exact solution.

Experimental Summary: We conduct extensive experiments on both real-world and synthetic datasets with a special emphasis on traffic matrix computation. We compare our method against a number of commonly used approaches such as an efficient quadratic programming based solver, a two stage approximate approach first proposed in [25] and one based on compressive sensing. Our experimental results show that our exact algorithm significantly outperforms the baselines. Furthermore, our threshold based approximation approaches inspired from similarity joins provide even more speedup over DIRECT without resulting in any significant increase in reconstruction error.

Summary of Contributions:

- We investigate the Signal Reconstruction Problem (SRP) which has diverse applications. By using techniques that were originally pioneered for databases, we dramatically improve the scale of problems that could be solved.
- We formulate SRP as a Quadratic Programming problem and derive its Lagrangian dual form and propose an exact algorithm DIRECT to solve the dual problem. Our algorithm DIRECT already outperforms commonly used approaches for SRP.
- We identify the computational bottleneck in DIRECT and propose a threshold-based algorithm for approximating it. We propose a hybrid algorithm that combines two algorithms that were designed for efficiently computing set similarity joins.
- We conduct a comprehensive set of experiments on both real and synthetic datasets that confirm the efficiency and effectiveness of our approach, and report the results in [1]. Here we provide a summary of those results.

Paper Organization: We provide the necessary background to SRP and formally define it in § 2. In § 3, we describe the exact algorithm DIRECT for solving SRP. In § 4, we show how to apply approximation using techniques from databases to significantly speed up the computation. In § 5, we discuss how our approach can be easily adapted to identify the top-K components of the reconstructed signal. § 6 describes our experiments followed by related work in § 7 with § 8 providing the conclusion.

2. PROBLEM FORMULATION

As mentioned in Section 1, we consider a special class of SRP that has a number of applications in network traffic engineering, tomographic image reconstruction and many others, discussed in § 2.1. We are given a system of linear equations $AX = b$ where

- $A \in \{1, 0\}^{n \times m}$ is a sparse binary matrix $n \ll m$.
- $X \in \mathbb{R}^m$ is the "signal" to be reconstructed and is a vector of unknown values.
- $b \in \mathbb{R}^n$ is the vector of observations.

Each row in the matrix A corresponds to an equation with each column corresponding to an unknown variable. When the number of equations (n) is much smaller than the number of unknowns (m), the system of linear equations is said to be under-determined and does not have a unique solution. The solution space can be represented as a hyperplane in a $m' \in [2, m]$ dimensional vector space[1]. Since SRP does not have a unique solution, one must have auxiliary criteria to choose the best solution from the set of (possibly infinite) valid solutions. A common approach in SRP is to provide a prior X' and the objective is to pick the solution X that is closest to X'. We study the problem where the objective is to find the point satisfying $AX = b$ that minimizes the ℓ_2 distance from a prior point X'. Formally the problem is defined as:

$$\begin{aligned} \min\ & \|X - X'\|_2 \\ \text{s.t.}\ & AX = b \end{aligned} \tag{1}$$

Figure 1 provides an example visualization of the problem in 3 dimensions. The gray plane is the solution space with the prior marked as a point X'. The intersection of the perpendicular line to the plane that passes though X' is the point that minimizes $\|X - X'\|_2$.

In this paper, we pay attention to the fact that SRP is a special case of quadratic programming where (a) the constraints are only in the form of equality, (b) matrix A is sparse, and (c) matrix A is binary (and hence un-weighted). By leveraging these characteristics, we seek to design more efficient solutions compared with the baselines that are designed for general cases. Especially, in § 3, after studying the existing work, we use the dual representative of the

[1]We assume that the problem has at least one solution.

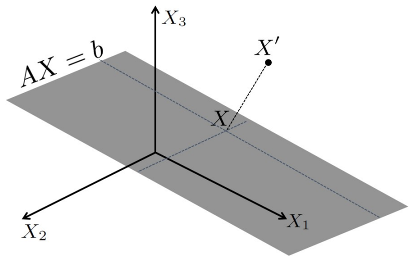


Figure 1: Visualizing the problem

problem to propose an efficient exact algorithm. Later in § 4, we show how leveraging similarity join techniques help in achieving significant speed up without sacrificing much accuracy.

2.1 Broad Application Range

SRP covers a broad range of real world problems that use signal reconstruction. In practice, it is popular to observe low-dimensional projections in form of (unweighted) aggregates of a high-dimensional signal vector. For example, in general network flow applications (such as road traffic estimation [27]), the value on each edge is the summation of the flows values which includes this edge as part of the path between them. Of course, a requirement to our problem is an "expert-provided" prior template, such as *gravity model* [25] for the network flow problems. Another major application domain for SRP problem over aggregates is image reconstruction (§ 2.1.2), where observations are unweighted projections of unknowns. Image reconstruction has a broad applications ranging from medical imaging [11, 14], to astronomy [23] and physics [19] and to gas flow reconstruction [2]. Some of the other applications of SRP, in general, include radar data reconstruction [18], transmission electron microscopy [15], and product assortment design [9], to name a few. To showcase some applications in more detail, we sketch a few examples in the context of network flow problems and image reconstruction in the following.

2.1.1 Network Tomography

Traffic matrix computation *(the running example)*: Consider an IP network with n traffic links and m source-destination traffic flows (SD flow) between the ingress/egress points, where $n \ll m$. The ingress/egress points can be PoPs (points of presence) or routers or even IP prefixes depending on the level of granularity required. The network has a routing policy prescribes a path for each of the SD flows that can be captured in a $\#links(n) \times \#flows(m)$ binary matrix A, where the entry $A[i, j] = 1$ if the link i is used to route the traffic of the j-th SD flow. The matrix A is sparse and "fat" with more SD flows(columns) than number of links(rows). Note that, one cannot directly measure each of the SD flows on a link owing to efficiency reasons. However, one can easily measure the total volume of the network traffic that passes through a given link using network protocols such as SNMP. Thus, the load on each link i becomes the observed vector b. To obtain a prior X', one can use any traffic model such as the popular and intuitive *gravity model* [25]. It assumes independence between source and destination and states that traffic between any given source s and destination d is proportional to the product of network traffic entering at s and that exiting at d.
Traffic analysis attack in P2P networks: In traffic analysis attack, the information leak on traffic data is exploited to expose the user traffic pattern in P2P networks [10]. Here we propose the following traffic analysis attack that can be modeled to our problem: Consider an adversary who monitors the link level traffics in a P2P network. Applying SRP, one can directly identify the volume of traffic between any pair of users in a P2P network.

2.1.2 Image Reconstruction

Image reconstruction [16,24] has a wide range of applications in different fields such as medical imaging [11, 14], and physics [19]. Given a set of (usually 2D) projection of a (usually 3D) image, the objective is to reconstruct it. The reconstruction is usually done with the help of some prior knowledge. For example, knowing that the 2D projections are taken from a human face, one may use a template 3D face photo and, among all possible 3D reconstructions from the 2D images, find the one that is the closest to the template, making the image reconstruction more effective.

CT Scan: A popular application of SRP is tomographic reconstruction, which is a multi-dimensional linear inverse problem with wide range of applications in medical imaging [11, 14] such as CT scans (computed tomography). Informally, a CT scan takes multiple 2D projections (b) through X-rays from different angles (A) and the objective is to reconstruct the 3D image from the projections. Note that many 3D images may produce the same projections necessitating the use of priors to choose an appropriate reconstruction.

Radio astronomy: In Astronomy, SRP has application for reconstructing interferometric images where the astrophysical signals are probed through Fourier measurements. The objective is to reconstruct the images from the observations – forming a SRP scenario. Also, the specific prior information about the signals plays an important role in reconstruction, as mentioned in [23].

3. EXACT SOLUTION FOR SOLVING SRP

In this section, we begin by describing two representative approaches for solving SRP from prior research and highlight their shortcomings. We then propose a dual representation of the problem that can be solved exactly in an efficient manner and already outperforms the baselines. This alternate formulation has a number of appealing properties that allows one to leverage various database techniques for speeding it up.

3.1 Lagrangian Formulation of SRP

In this subsection, we leverage the Lagrangian dual form of SRP as a special case of quadratic programming, and design an efficient exact solution for it. For SRP as specified in Equation 1, $f(X) = \frac{1}{2}X^TX - X'^TX$ and $g(X) = AX$.[2] Thus, our problem can be re-written as:

$$L(X, \lambda) = \frac{1}{2}X^TX - X'^TX + \lambda^T(AX - b) \qquad (2)$$

Next, we find the stationary point [3] of Equation 2 in the general form by taking the derivatives with regard to X and λ and setting them to zero, we get:

$$X = X' - A^T(AA^T)^{-1}(AX' - b) \qquad (3)$$

Solving SRP in Dual Form. The stationary point of Equation 2 is the optimal solution for our problem (Equation 1). In contrast to prior work, we solve the SRP problem by directly solving Equation 3. We make two observations. First, the matrix $AA^T \in \mathbb{Z}^{n\times n}$ always has an inverse as it is full-rank. From Figure 1, one can note that the problem has a unique solution that minimizes the distance from the prior. It means that AA^T is full-rank, because otherwise the problem was not feasible and would not have a solution. Second, Equation 3 does have a matrix inverse operator that is expensive to compute. However, one can avoid taking the inverse of AA^T

[2]Note that $\min \frac{1}{2}X^TX - X'^TX$ is the same as $\min ||X - X'||_2$.

[3]Since, looking at Figure 1, Equation 1 has a single optimal point, Equation 2 has one stationary point which happens to be the saddle point.

0	0	0	1	0	0	0	1	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	1	0	1	0	1
0	1	0	0	0	0	1	0	0	0

(a)

$\langle 3, 7 \rangle$
$\langle 2 \rangle$
$\langle 5, 7, 9 \rangle$
$\langle 1, 6 \rangle$

(b)

Figure 2: Illustration of the sparse representation of A. (a) Non sparse representation, (b) Sparse representation

by computing ξ in Equation 4, and replacing $(AA^T)^{-1}(AX' - b)$ by it in Equation 3.

$$(AA^T)\xi = AX' - b \qquad (4)$$

Algorithm 1 provides the pseudocode for DIRECT.

Algorithm 1 DIRECT
Input: A, b, and X'
Output: X

```
1: $t = AA^T$
2: $t_2 = AX' - b$
3: Solve system of linear equations: $t\,\xi = t_2$
4: $X = X' - A^T\xi$
5: return $X$
```

Performance Analysis of DIRECT**.** Let us now investigate the performance of our algorithm. Recall that A is a fat matrix with $n \ll m$ while X and X' are m-dimensional vectors, and b is a n-dimensional vector. Line 1 of Algorithm 1 takes $O(n^2m)$ while Line 2 takes $O(nm)$. Line 3 involves solving a system of linear equations. A naive way would be to compute the inverse of t that can take as much as $O(n^3)$. However, by observing that t is sparse, one can use approaches such as Gauss-Jordan elimination or other iterative methods that are practically much faster for sparse matrices. Finally, the computation of Line 4 is in $O(nm)$. Looking at DIRECT holistically, one can notice that its computational bottleneck is Line 1 thereby making the overall complexity to be $O(n^2m)$.

An additional approach to speedup DIRECT is to observe that matrix A is sparse and thereby store it in a manner that allows efficient matrix multiplication. Since A is binary (and hence unweighted), a natural representation is to store only the indices of non-zero values. Figures 2a and 2b show the non-sparse and sparse representation of a matrix A. Note that AA^T is symmetric since $t[i, j]$ and $t[j, i]$ are obtained by the dot product of rows i and j of A. Let l be the number of non-zero elements in each row. Since A is sparse, $l \ll m$, one can design a natural matrix multiplication algorithm with time complexity of $O(nml)$ that is orders of magnitude faster than algorithm such as Strassen algorithm.

4. TRADING OFF ACCURACY WITH EFFICIENCY

In many applications of SRP, m is often in $O(n^2)$, thereby making the computational complexity of DIRECT to be $O(n^4)$. The key bottleneck is the computation of AA^T. On the other hand, for large problem instances, the user may accept trading off accuracy with efficiency and prefer a close-to-exact solution that is computed quickly, rather than the expensive exact solution. In this section, our objective is to speed up DIRECT by computing the bottle-neck step, i.e., computing AA^T, approximately. We show how to leverage a threshold-based approach by only computing the values of matrix AA^T that are larger than a certain threshold. We describe the connection between this problem variant and similarity joins and propose a hybrid method by adopting two classical algorithms designed for similarity estimation, which results in an efficient solution for computing AA^T.

4.1 Bounding Values in Matrix AA^T

We begin by showing that one can efficiently compute the bound for each cell value in matrix AA^T. Figure 3 shows a sparse matrix A with 183 rows and 495 columns, in which the non-zero elements are highlighted in white. Figure 4 shows the non-zero elements in matrix AA^T. We can notice that AA^T is square and also sparse due to the fact that every element of AA^T is the dot product of two sparse vectors (two rows of matrix A). Furthermore, one can also observe a more subtle phenomenon that we state in Theorem 1 which could used to design an efficient algorithm.

THEOREM 1. *Given a sparse binary matrix A, considering the elements on the diagonal of AA^T, i.e., $t[i, i]$, $\forall 0 \le i < n$:*

- *$t[i, i] = |A[i]|$, where $|A[i]|$ is the number of non-zero elements in row $A[i]$.*
- *$t[i, i]$ is an upper bound for the elements in the row $t[i]$ and the column $t[, i]$; formally, $\forall 0 \le j < n : t[i, j] \le t[i, i]$ and $t[i, j] \le t[j, j]$.*

The proof can be found in [1].

Consider two representations of AA^T of the example matrix given in Figure 3. Figure 4 shows all the non-zero elements of AA^T while Figure 5 shows a magnitude-weighted variant wherein cells with larger values are plotted in brighter colors. Figure 5 visually shows that the elements on the diagonal are brighter than the ones in the same row and column as predicted by Theorem 1. Furthermore, one may notice that most of the non-zero elements of AA^T (in Figure 4) are small values (in Figure 5). This shows that while there are a reasonable number of non-zero elements, the number of elements with higher magnitude is often much smaller. Next, we use this insight along with Theorem 1 for speeding up DIRECT.

4.2 Threshold Based Computation Of AA^T

In the previous subsection, we discussed the bound on the cell values in AA^T and showed that a small number of elements in AA^T take the bulk of the value. This is the key in designing a threshold-based algorithm for computing AA^T wherein we only compute values of AA^T that are above a certain threshold. Specifically, we use the elements on the diagonal as an upper-bound and only compute the elements for which this upper-bound is larger than a user-specified threshold. Note that, if the threshold is equal to 1, the algorithm will compute the values of all elements. However, the user-specified threshold allows additional opportunities for efficiency.

Algorithm 2 provides the pseudocode for the threshold-based multiplication of sparse binary matrix A with its transpose. This algorithm depends on the existence of an oracle called SIM that given two rows $A[i]$ and $A[j]$, and the threshold τ, returns the dot product of $A[i]$ and $A[j]$ if the result is not less than τ.

4.3 Leveraging Similarity Joins for Oracle SIM

The database community has extensively studied mechanisms for computing set similarity for applications such as data cleaning [7] where the objective is to efficiently identify the set of tuples that are "close enough" on multiple attributes. In this subsection, we describe how to implement the oracle SIM by leveraging prior research on computing set similarity. Especially, we propose a hybrid method that combines the threshold-based similarity joins with the sketch-based methods to resolve their shortcomings.

Oracle SIM through Set Similarity. Given two rows $A[i]$ and $A[j]$, and the threshold τ, SIM should find the dot product of $A[i]$

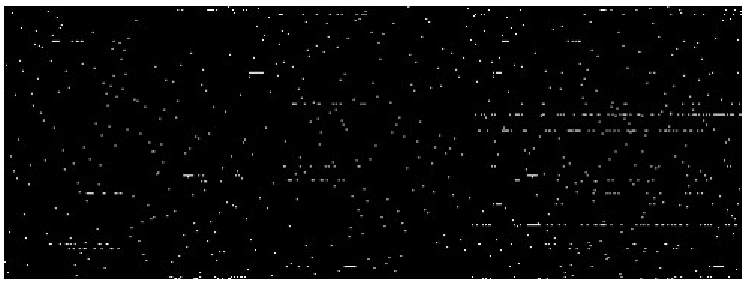


Figure 3: An example of the binary sparse matrix $A_{183\times 495}$

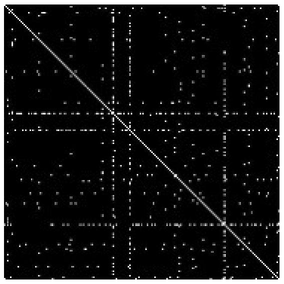


Figure 4: The non-zero elements in AA^T for the example of Figure 3

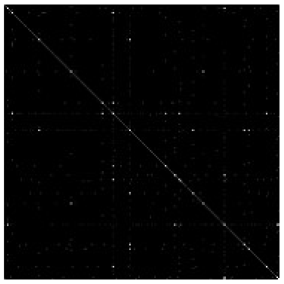


Figure 5: Magnitude of weights in AA^T for the example of Figure 3

```
Algorithm 2 Approx$AA^T$
Input: Sparse matrix $A$, Threshold $\tau$
Output: $t$
1: $\mathcal{F} = \{\}$
2: for $i = 0$ to $n-1$ do
3:     $t[i,i] = |A[i]|$
4:     if $|A[i]| \geq \tau$ then add $i$ to $\mathcal{F}$
5: end for
6: for every pair $i, j \in \mathcal{F}$ do
7:     $t[i,j] = t[j,i] = \text{SIM}(A[i], A[j], \tau)$
8: end for
9: return $t$
```

and $A[j]$ if it is not less than τ. It is possible to make an interesting connection between SIM and sets similarity problems as follows. Let every column in matrix A be an object o in a universe $\mathcal{U}$ of m elements. Every row $A[i]$ represents a set U_i in $\mathcal{U}$, where $\forall o_j \in \mathcal{U}$, $o_j \in U_i$ *iff* $A[i,j] = 1$. Equivalently, each row corresponds to a set U_i that stores the indices of the non-zero columns similar to Figure 2b. Using this transformation, we can see that our objective is to compute $|U_i \cap U_j|$ for all pairs of sets U_i and U_j where $|U_i \cap U_j| \geq \tau$. Note that we represent $|U_i \cap U_j|$ by $\cap_{i,j}$ and $|U_i \cup U_j|$ by $\cup_{i,j}$ respectively.

Due to its widespread importance, different versions of this problem have been extensively studied in the DB community. In this paper, we consider one exact approach and two approximate approaches based on threshold-based algorithms [7] and sketch-based methods [3, 8, 12]. We then compare and contrast the two approximate approaches, describe the scenarios when they provide better performance, and propose a hybrid algorithm based on these scenarios.

Exact Approach : Set Intersection. One can see that when $\tau = 1$, the problem boils down to computing AA^T exactly. This in turn, boils down to computing the intersection between two sets as efficiently as possible. The sparse representation of the matrix often provides the non-zero columns in an ordered manner. The simplest approaches for finding the intersection of ordered sets is to perform a linear merge by scanning both the lists in parallel and leveraging the ordered nature similar to the merge step of merge-sort. One can also speedup this approach by using sophisticated approaches such as binary search on one of the lists or using sophisticated data structures such as treaps or skip-lists. Each of these approaches allows one to "skip" some elements of a set when necessary.

Approximate Approach : Threshold based Algorithms. Threshold-based algorithms, such as [7] identify the pair of sets such that their similarity is more than a given threshold. This has a number of applications such as data cleaning, deduplication, collaborative filtering, and product recommendation in advertisement where the objective is to quickly identify the pairs that are highly similar. The key idea is that if the intersection of two sets is large, the intersection of small subsets of them is non zero [7]. More precisely, for two sets U_i and U_j with size h, if $\cap_{i,j} \geq \tau$, any subsets $U'_i \subset U_i$ and $U'_j \subset U_j$ of size $h - \tau + 1$ will overlap; i.e., $|U'_i \cap U'_j| > 0$. Using this idea, while considering an ordering of the objects, the algorithm first finds the set of candidate pairs that overlap in a subset of size $h - \tau + 1$. In the second step, the algorithm verifies the pairs, by removing the false positives.

One can see the effectiveness of this method highly depends on the value of τ and, considering the target application, it works well for the cases that τ is large. For example, consider a case where $h = 100$. When $\tau = 99$ (i.e., 99% similarity), the first filtering step needs to compare the subsets of size 2 and is efficient; whereas if $\tau = 10$, the filtering step needs to compare the subset pairs of size 91, which is close to the entire set. The latrer case is quite possible in our problem. To understand it better, let us consider matrix A in Figure 3, while setting τ equal to 5 in Algorithm 2. Even though the size of many of the rows is close to the threshold, there are rows $A[i]$ where $|A[i]|$ is significantly larger than it. For example, for two rows $A[i]$ and $A[j]$ where $|A[i]| \geq 50$ and $|A[j]| \geq 50$, to satisfy the dot product be not less than τ, the filtering step needs to compare the subsets of size ≥ 44, which is close to the exact comparison of $A[i]$ and $A[j]$.

Approximate Approach : Sketch based Algorithms. Sketch based methods such as [3, 8, 12] use a precomputed synopsis such as a minhash for answering different set aggregates such as Jaccard similarity. The main idea behind the min-hashing [4] based algorithms is as follows: consider a hash (ordering) of the elements in $\mathcal{U}$. For each set U_i, let $h_{\min}(U_i)$ be the element $o \in U_i$ that has the minimum hash value. Two sets U_i and U_j have the same min-hash, when the element with the smallest hash value belongs to their intersection. Hence, it is easy to see that the probability that $h_{\min}(U_i) = h_{\min}(U_j)$ is equal to $\frac{\cap_{i,j}}{\cup_{i,j}}$, i.e., Jaccard similarity of U_i and U_j. Bottom-k sketch [8], a variant of min-hashing picks the hash of the k elements in U_i with the smallest hash value, as its signature. The Jaccard similarity of two sets U_i and U_j is estimated as $\frac{k_\cap(i,j)}{k}$, where $k_\cap(i,j)$ is $|h_k(U_i) \cap h_k(U_j)|$. Bayer et al. [3] use the bottom-k sketch for estimating the union and intersection of the sets. Let $h_{i,j}[k]$ be the hash value of the k-th smallest hash value in $h_k(U_i) \cup h_k(U_j)$. The idea is that the larger the size of a set is, the smaller the expected value of the k-th element in hash is. Using the results of [3], $\frac{m(k-1)}{h_{i,j}[k]}$ is an unbiased estimator for $\cup_{i,j}$. Hence the estimation for $\cap_{i,j}$ is as provided in Equation 5.

$$E[\cap_{i,j}] = \frac{k_\cap(i,j)}{k} \frac{m(k-1)}{h_{i,j}[k]} \tag{5}$$

Estimating $\cup_{i,j}$ with Equation 5, performs well when $\cup_{i,j} \gg$

1 [3], i.e., the larger sets. Hence, we combine the threshold-based and sketch-based algorithms to design the oracle SIM, as a hybrid method that, based on the sizes of the rows $A[i]$ and $A[j]$, adopts the threshold-based computation with sketch-based estimation for computing the dot product of $A[i]$ and $A[j]$. We consider $\log(m)$ as the threshold to decide which strategy to adopt. Considering the effectiveness of threshold based approaches when U_i and U_j are small and, as a result, the two sets need a large overlap to have the intersection larger than τ, if $|U_i|$ and $|U_j|$ are less than $\log(m)$, we choose the threshold-based intersection computation. However, if the size of U_i or U_j is more then we use the bottom-k sketch, while considering k to be $\log(m)$. For each element $o_j \in \mathbb{U}$, we set $h(o_j) = j$. Hence, for each vector U_i the index of the first $\log(m)$ elements in it are its bottom-k sketch. Using this strategy, Algorithm 3 shows the pseudo code of the oracle SIM.

Given two given sets U_i and U_j (corresponding to the rows $A[i]$ and $A[j]$) together with the threshold τ, the algorithm aims to compute the value of $\cap_{i,j}$, if it is larger than τ. Combining the two aforementioned methods, if $|U_i|$ and $|U_j|$ are more than a value α, the algorithm uses sampling to estimate $\cap_{i,j}$, otherwise it applies the threshold-based method to compute it. During the sampling, rather than sampling from $\mathcal{U}$, the algorithm samples from U_i to reduce the underestimation of probability. In this case, in order to compute $\cap_{i,j}$, the algorithm, for each sample, picks a random object from U_i and check its existence in U_j. It is easy to see it is an unbiased estimator for $\cap_{i,j}$, where its expected value is $\cap_{i,j}$. If $|U_i|$ or $|U_j|$ is less than α, the algorithms applies threshold-based strategy for computing $\cap_{i,j}$. As discussed earlier in this subsection, in order for $\cap_{i,j}$ to be more than τ, the subsets of size $\cap_{i,j} - \tau + 1$ should intersect. Hence, the algorithm first applies the threshold filtering and only if the two subsets intersect it continues with computing $\cap_{i,j}$.

Algorithm 3 SIM
Input: the sets U_i and U_j, Threshold τ
Output: c

```
1:  if $|U_i| \geq \log(m)$ and $|U_j| \geq \log(m)$ then
2:      $h_i$ = the first $k$ elements in $U_i$
3:      $h_j$ = the first $k$ elements in $U_j$
4:      $k_\cap(i,j) = |h_i \cap h_j|$
5:      $h_{i,j}[k]$ = the first $k$ elements in $h_i \cup h_j$
6:      $c = \frac{k_\cap(i,j)}{k} \frac{m(k-1)}{h_{i,j}[k]}$
7:  else
8:      $c = 0$
9:      if $|U_i| > |U_j|$ then swap $U_i$ and $U_j$
10:     $\beta = |U_i| - \tau$
11:     for $k = 0$ to $\beta$ do: if $U_i[k] \in U_j$ then $c = c + 1$
12:     if $c = 0$ then return 0
13:     for $k = \beta$ to $|U_i| - 1$ do: if $U_i[k] \in U_j$ then $c = c + 1$
14: end if
15: return $c$
```

Performance Analysis. Algorithm 2 has a time complexity of $O(n + \mu^2 \min(l, \log(m)))$, where $\mu = |\{A[i]| \, |A[i]| \geq \tau\}|$.

5. SCALING SRP TO VERY LARGE SETTINGS

Recall that in SRP often n is a low dimensional vector with $n \ll m$. In this subsection we briefly describe how to extend DIRECT to handle cases where even n is very large (and still $n \ll m$). For example, let n be 10^6 and m be 10^{12}. A key aspect of DIRECT is that it leverages the sparse representation of the matrix (as against its complete dense representation) for speedup. However, when n is very large, even fitting the sparse representation of A into the memory may not be possible. To see why, even if there is only one non-zero value in *every column*, then we use $O(m)$ storage to even represent this matrix.

Interestingly, the similarity-joins based techniques proposed in § 4 do not require to completely materialize even sparse representation of A for estimating AA^T. Also, there are many scenarios where the user is interested in knowing the values of a subset of components of the reconstructed signals such as those corresponding to the largest values of the reconstructed signal. We now show how to adapt our algorithms to handle these scenarios.

Consider Algorithm 1 where the critical step is the first line. Algorithm 3 applies bottom-k sketch for the sets whose size is more than $\log m$. Thus, choosing the signature size in the bottom-k sketch to be in $O(\log m)$, Algorithm 3 needs at most $O(\log m)$ elements from *each row*. As a result, Line 1 of DIRECT needs a representation of size $O(n \log m)$ of A. For instance, in our example of $n = 10^6$ and $m = 10^{12}$, the size of the representative of A is only in the order of 1 million rows by 40 columns. Also, since AA^T is a sparse matrix, we only store the non-zero values of matrix t, rather than the complete n by n matrix. Line 2 is the multiplication of matrix A with X' whose dimensions are m by 1 followed by subtracting the n-dimensional result vector from the vector b. For this line, for each row of A, we use a sample of size $O(\log m)$ for the non-zero elements of the row, while using the values of X' as the sampling distribution. The result is a representation of size $O(n \log m)$ of A. Also, rather than loading the complete vector X' to the memory, in an iterative manner, we bring loadable buckets of it to the memory, update the calculation for that bucket, and move to the next one. In Line 4, t is the non-zero elements of AA^T and t' is a n by 1 vector, and finding the n by 1 vector ξ is doable, using methods like Gauss-Jordan. Finally, we only limit the calculations to the variables of interest, or even if the computation of all variables is required, in an iterative manner, we move a loadable bucket of them to the memory, compute their values, and move to the next bucket.

6. EXPERIMENTAL EVALUATION

6.1 Experimental Setup

Hardware and Platform. All our experiments were performed on a Macintosh machine with a 2.6 GHz CPU and 8GB memory. The algorithms were implemented using Python2.7 and Matlab.

Datasets. We conducted extensive experiments to demonstrate the efficacy of our algorithms over graphs with diverse values for number of nodes, edges and source-destination pairs. Recall that given a communication network, the size of the routing matrix A is parameterized by the number of edges and number of source-destination pairs - and not by the number of nodes and edges. The size of SRP that we tackle are 2-3 orders of magnitude larger than prior work such as [26]. Specifically, we used p2p dataset from SNAP repository of Stanford university[4]. The p2p dataset is a snapshot of the Gnutella network in August 2002 with 10876 nodes and 39994 edges. Nodes represent the hosts and the links represent the connection between the hosts. Each of the derived datasets is a subgraph of the overall p2p graph and was obtained by Forest Fire model [17]. The characteristics of each of these datasets dubbed *p2p-2* and *p2p-3* can be found in Table 1.

Constructing Traffic Matrices. Once we sample the network and obtain a connected graph, we consider all possible source destination pairs, i.e., #nodes×(#nodes−1), to be as individual flows.

[4]SNAP Dataset: https://snap.stanford.edu/data/p2p-Gnutella04.html

Table 1: Dataset Characteristics

Network	#Nodes	#Edges	#Source-Destination pairs
N_1	274	281	827
p2p-3	1438	7081	2M

For each source-destination pair we calculated the shortest path between them (network policies are not considered here as our algorithm is oblivious of the route chosen). Traffic matrix is a collection of all such routes in the following manner, each of the rows corresponds to an edge used in routing and each of the columns corresponds to a source-destination pair. Every cell, $c[i, j]$ is a '1' if edge$[i]$ is involved in routing traffic for source-destination$[j]$ else is assigned a value '0'. A visual glimpse of the routing matrix is given in Figure 2.

We used a *Pareto* traffic generation model, a popular stochastic model of the traffic flows for generating self-similar traffic observed in network communication [6]. The distribution is parametrized by a scale parameter x_m (set to 20) and a shape parameter α (set to 1). x_m is the minimum value of the distribution of traffic represented by the scale parameter while the shape parameter α indicates the 'steepness of the slope' of the distribution curve. The *prior* to the experiments (X') was obtained as a function of gravity model from [25].

6.2 Experimental Results

We compare the exact algorithm DIRECT with the baselines QP and WLSE [25]. The evaluation was conducted over small scale synthetic networks. Here we report the comparison results for N_1 (Table 1). Please refer to [1] for the complete experiment results. As shown in Figure 6, DIRECT significantly outperforms the baselines. In addition to comparing with these two baselines, for N_1, we also used compressive sensing [20] for estimating the values of the source-destination pairs. Since the objective in compressive sensing is the expensive l_0-optimization, even for our smallest setting N_1 it took 23.414 seconds.

We next evaluate the exact version of DIRECT and its approximate counterpart (using Algorithm 2) that leverages techniques from similarity joins to speed up the computation. We use DIRECT-E to refer to the exact version of DIRECT and DIRECT-A for its approximate version. Note that our algorithms take advantage of the sparse representation of matrix A and can perform the linear algebraic operations without materializing the entire matrix. We also evaluate the performance of our algorithms to two different threshold values of $(m/1000)$ and $(m/100)$, where m is the number of source-destination pairs. Choosing an appropriate threshold is often domain specific with larger thresholds providing better speedups. We compare the performance of the algorithms DIRECT-E and DIRECT-A through two metrics : performance and accuracy. We measure the former through execution time. We measure the accuracy of the signal reconstruction through *bucketized error* where we bucketize the source-destination pairs by the exact value of their flows and compute the error of the approximation algorithm within each bucket. The bucketization is often more illuminating for scenarios such as network traffic engineering where the signal exhibits a heavy tailed distribution and often the practitioner is interested in accurately estimating large flows. After finding the optimal flow assignments using the algorithm DIRECT-E, we sort the source-destination pairs in descending order, based on the amount of flow passing through them. For example, let a flow assignment by DIRECT-E be $\{(SD_1 : 3), (SD_2 : 24), (SD_3 : 7), (SD_4 : 75), (SD_5 : 5), (SD_6 : 12)\}$. The sorted SD pairs are $\{(SD_4 : 75), (SD_2 : 24), (SD_6 : 12), (SD_3 : 7), (SD_5 : 5), (SD_1 : 3)\}$. We then partition the SD pairs into 50 equal size buckets (each bucket contains 2% of SD pairs[5]). In the provided example, assume that we partition them into 3 buckets $B_1 : \{(SD_4 : 75), (SD_2 : 24)\}$, $B_2 : \{(SD_6 : 12), (SD_3 : 7)\}$, and $B_3 : \{(SD_5 : 5), (SD_1 : 3)\}$. For every SD pair, we consider the difference between the values computed by DIRECT-A and the one by DIRECT-E as the error of that SD pair, and compute the average for each bucket. In our example, let $\{(SD_1 : 5), (SD_2 : 24), (SD_3 : 6), (SD_4 : 79), (SD_5 : 5), (SD_6 : 11)\}$ be the assigned values by DIRECT-A. Then the average errors for the buckets B_1, B_2, and B_3 are 2, 1, and 1, respectively. It was observed in [25] that for many tasks in network traffic engineering such as routing optimization, even a relative error of *few* 10s of percent is considered tolerable.

p2p-3 (2M Source-Destination pairs) This network has 2M source-destination pairs with 7081 edges sampled from the *SNAP p2p* dataset. Figure 7 shows that DIRECT-E takes much as 1500 seconds to compute the exact solution. This is often prohibitive and simply unacceptable for many traffic engineering tasks. However, our approximate algorithms can provide the result in as little as 35 seconds. This is a significant reduction in execution time with a speedup of much as 97% of the running time of DIRECT-E. Figure 8 shows that the results are very close to the exact answer produced by DIRECT-E.

7. RELATED WORK

Linear Algebraic Techniques for Solving SRP: There has been extensive work on solving the system of linear equations using a wide variety of techniques such as computing the pseudoinverse of A [22] or performing Singular Value Decomposition (SVD) on A, and iterative algorithms for solving the linear system [22]. However, none of these methods scale for large-scale signal reconstruction problems. A key bottleneck in these approaches is often the computation of the pseudo inverse for matrix A. Note that any matrix B such that $ABA = A$ is defined as a pseudo inverse for A. It is possible to identify "the infinitely many possible generalized inverses" [22], each with its own advantages and disadvantages. Moore-Penrose Pseudo inverse (MPP) [21] is one of the most well-known and widely used pseudo inverse. MPP is the pseudo inverse that has the smallest Frobenius norm, minimizes the least-square fit in over-determined systems, and finds the shortest solution in the under-determined ones. However, none of the pseudo-inverse definitions suits our purpose of finding the solution X that minimizes the ℓ_2 distance from a prior. Furthermore, computing pseudo inverses is often done by SVD that is computationally very expensive.

8. CONCLUSION

In this paper, we investigated how a wide ranging problem of large scale signal reconstruction can benefit from techniques developed by the database community. Efficiently solving SRP has number of applications in diverse domains including network traffic engineering, astronomy, medical imaging etc. We propose an algorithm DIRECT based on the Lagrangian dual form of SRP. We identify a number of computational bottlenecks in DIRECT and evaluate the use of database techniques such as sampling and similarity joins for speeding them up without much loss in accuracy. Our experiments on networks that are orders of magnitude larger than prior work show the potential of our approach.

[5]We have found out the knee point of the cumulative flow is around 2%.

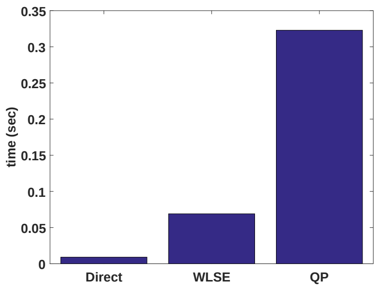


Figure 6: Direct **v.s. baselines in** $N_1 : n = 281$ **and** $m = 827$

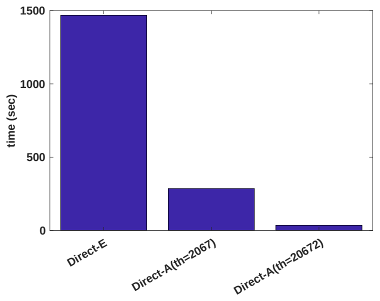


Figure 7: Execution time of Direct-e, Direct-a (τ=2067)**, and** Direct-a (τ=20672) **in p2p-3**

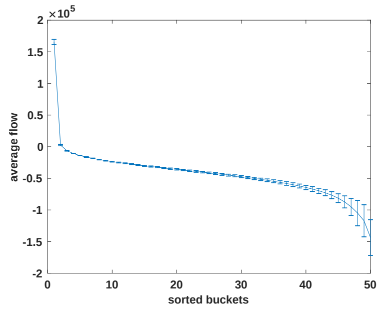


Figure 8: Absolute Error of the Direct-a (τ = 20672) **in p2p-3**

9. ACKNOWLEDGMENTS

The work of Abolfazl Asudeh, Azade Nazi, Jees Augustine, and Gautam Das was supported in part by AT&T, the National Science Foundation under grant 1343976, and the Army Research Office under grant W911NF-15-1-0020. Nan Zhang was supported in part by the National Science Foundation, including under grants 1343976, 1443858, 1624074, 1760059, and by the Army Research Office under grant W911NF-15-1-0020.

10. REFERENCES

[1] A. Asudeh, A. Nazi, J. Augustine, S. Thirumuruganathan, N. Zhang, G. Das, and D. Srivastava. Leveraging similarity joins for signal reconstruction. *PVLDB*, 11(10), 2018.

[2] Y. Awatsuji, Y. Wang, P. Xia, and O. Matoba. 3d image reconstruction of transparent gas flow by parallel phase-shifting digital holography. In *WIO*, 2016.

[3] K. Beyer, R. Gemulla, P. J. Haas, B. Reinwald, and Y. Sismanis. Distinct-value synopses for multiset operations. *Communications of the ACM*, 52(10):87–95, 2009.

[4] A. Z. Broder. On the resemblance and containment of documents. In *SEQUENCES*, pages 21–29. IEEE, 1997.

[5] J. Cao, D. Davis, S. Vander Wiel, and B. Yu. Time-varying network tomography: router link data. *Journal of the American statistical association*, 95(452):1063–1075, 2000.

[6] B. Chandrasekaran. Survey of network traffic models. *Waschington University in St. Louis CSE*, 567, 2009.

[7] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*. IEEE, 2006.

[8] E. Cohen and H. Kaplan. Tighter estimation using bottom k sketches. *PVLDB*, 1(1):213–224, 2008.

[9] V. F. Farias, S. Jagabathula, and D. Shah. A nonparametric approach to modeling choice with limited data. *Management science*, 59(2):305–322, 2013.

[10] Y. Gong. Identifying p2p users using traffic analysis. 2005. `www.symantec.com/connect/articles/identifying-p2p-users-using-traffic-analysis`.

[11] P. Grangeat and J.-L. Amans. *Three-dimensional image reconstruction in radiology and nuclear medicine*, volume 4. Springer Science & Business Media, 2013.

[12] M. Hadjieleftheriou, X. Yu, N. Koudas, and D. Srivastava. Hashed samples: selectivity estimators for set similarity selection queries. *PVLDB*, 1(1):201–212, 2008.

[13] P. C. Hansen. *Rank-deficient and discrete ill-posed problems: numerical aspects of linear inversion*. SIAM, 1998.

[14] W. T. Hrinivich, D. A. Hoover, K. Surry, C. Edirisinghe, D. D'Souza, A. Fenster, and E. Wong. Ultrasound guided high-dose-rate prostate brachytherapy: Live needle segmentation and 3d image reconstruction using the sagittal transducer. *Brachytherapy*, 15:S195, 2016.

[15] S. V. Kalinin, E. Strelcov, A. Belianinov, S. Somnath, R. K. Vasudevan, E. J. Lingerfelt, R. K. Archibald, C. Chen, R. Proksch, N. Laanait, et al. Big, deep, and smart data in scanning probe microscopy, 2016.

[16] P. Kuchment and F. Terzioglu. 3d image reconstruction from compton camera data. *arXiv:1604.03805*, 2016.

[17] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *SIGKDD*, pages 177–187. ACM, 2005.

[18] Z. Liu, Z. Shi, M. Jiang, J. Zhang, L. Chen, T. Zhang, and G. Liu. Using MC algorithm to implement 3d image reconstruction for yunnan weather radar data. *Journal of Computer and Communications*, 5(05), 2017.

[19] R. Massey, J. Rhodes, R. Ellis, N. Scoville, A. Leauthaud, A. Finoguenov, P. Capak, D. Bacon, H. Aussel, J.-P. Kneib, et al. Dark matter maps reveal cosmic scaffolding. *arXiv preprint astro-ph/0701594*, 2007.

[20] D. Needell and J. A. Tropp. Cosamp: Iterative signal recovery from incomplete and inaccurate samples. *Applied and Computational Harmonic Analysis*, 26(3), 2009.

[21] R. Penrose. A generalized inverse for matrices. In *Mathematical proceedings of the Cambridge philosophical society*, volume 51, pages 406–413, 1955.

[22] C. R. Vogel. *Computational methods for inverse problems*. SIAM, 2002.

[23] Y. Wiaux, L. Jacques, G. Puy, A. M. Scaife, and P. Vandergheynst. Compressed sensing imaging techniques for radio interferometry. *Monthly Notices of the Royal Astronomical Society*, 395(3):1733–1742, 2009.

[24] G. L. Zeng. 3d image reconstruction. In *Medical Image Reconstruction*, pages 87–123. Springer, 2010.

[25] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg. Fast accurate computation of large-scale ip traffic matrices from link loads. In *SIGMETRICS*, volume 31, 2003.

[26] Y. Zhang, M. Roughan, C. Lund, and D. Donoho. An information-theoretic approach to traffic matrix estimation. In *SIGCOMM*, pages 301–312. ACM, 2003.

[27] Y. Zhu, Z. Li, H. Zhu, M. Li, and Q. Zhang. A compressive sensing approach to urban traffic estimation with probe vehicles. *IEEE Transactions on Mobile Computing*, 12(11):2289–2302, 2012.

Technical Perspective: How Do Humans and Data Systems Establish a Common Query Language?

H. V. Jagadish
University of Michigan
jag@umich.edu

We all structure information in our brains: without structure, we would not be able to deal with the huge quantities of highly heterogenous information we process. However, each of us structures this information slightly differently, often leading to misunderstandings or requiring additional rounds of dialog to clarify. Database schema are also designed by humans. The structure imposed on the information by the schema reflects the human designers' perspective on the world, even if mediated through formal design techniques or computer software. Therefore, the structured data querying task can be viewed as having a schema mapping problem at its core: mapping between the "schema" the human has in her brain and the schema used to organize the database.

In short, database querying would be much easier if only human users could know exactly how the database was structured (and also what sort of data it contained). Indeed, there is a significant body of work on data exploration. The basic idea here is that the human learns about what is in the database and how it is structured, leading her to the data items of interest. In the process, the user typically specifies a sequence of (exploratory) queries, each based in part on the knowledge about the database that the user has gained thus far. The system attempts to facilitate this exploration by the user. See, for example, [1], for a review of such work.

A completely distinct stream of work deals with approximately specified queries. The user provides a query intent, whether in a query language with wild cards, in natural language, or by example; the system then works hard to understand (and execute) the intended query. In the process, the user's incomplete specification is completed, any errors in it are corrected, and so on. In some proposals, the system may even engage in dialog with the user to clarify user intent. However, the assumption is that the user intent is fixed during this process.

Seeing these two very distinct bodies of work on making databases easier to query, one should take a step back and see that in both cases, the system and the user have a shared objective of the system providing the user with the desired information. In the former, the user works to learn about the database; in the latter, the system works to learn about user intent. This paper bridges the two approaches and asks why both couldn't learn about each other.

Of course, this sort of two-sided learning is easier said than done. Figuring out how to do this effectively is the heart of the technical content in this paper. The authors model it as a cooperative two-player game, where both the user and the system are trying to achieve the same objective, which is to satisfy the user's information need. In each round of the game, the user may specify the need differently to help the system get to the right answer, and the system may interpret the user query differently, based on its growing understanding of the user's need. The strategy to play this game is learned through reinforcement learning.

This paper will open a whole new line of research, in which the user query statement is not kept fixed even if their information need remains unchanged.

1. REFERENCES

[1] Matteo Lissandrini, Davide Mottin, Themis Palpanas, and Yannis Velegrakis, 2018. Data Exploration Using Example-Based Methods. *In Synthesis Lectures on Data Management*, Morgan and Claypool. DOI= (https://doi.org/10.2200/S00881ED1V01Y201810DTM053)

.

How Do Humans and Data Systems Establish a Common Query Language?

Ben McCamish
Oregon State University
mccamisb@oregonstate.edu

Vahid Ghadakchi
Oregon State University
ghadakcv@oregonstate.edu

Arash Termehchy
Oregon State University
termehca@oregonstate.edu

Liang Huang
Oregon State University
liang.huang@oregonstate.edu

Behrouz Touri
University of CA San Diego
btouri@eng.ucsd.edu

ABSTRACT

As most users do *not* precisely know the structure and/or the content of databases, their queries do *not* exactly reflect their information needs. While database management systems (DBMS) may interact with users and use their feedback on the returned results to learn the information needs behind their queries, current query interfaces assume that users do *not* learn and modify the way way they express their information needs in form of queries during their interaction with the DBMS. Using a real-world interaction workload, we show that users learn and modify how to express their information needs during their interactions with the DBMS and their learning is accurately modeled by a well-known reinforcement learning mechanism. As current data interaction systems assume that users do *not* modify their strategies, they cannot discover the information needs behind users' queries effectively. We model the interaction between users and DBMS as a game with identical interest between two rational agents whose goal is to establish a common language for representing information needs in form of queries. We propose a reinforcement learning method that learns and answers the information needs behind queries and adapts to the changes in users' strategies and prove that it stochastically improves the effectiveness of answering queries. We propose two efficient implementation of this method over large relational databases. Our empirical studies over real-world query workloads indicate that our algorithms are efficient and effective.

1. INTRODUCTION

Most users do not know the structure and content of databases and concepts such as schema or formal query languages sufficiently well to express their information needs precisely in the form of queries [8]. They may convey their intents in easy-to-use but inherently ambiguous forms, such as keyword queries, which are open to numerous interpretations. Thus, it is very challenging for a database management system (DBMS) to understand and satisfy the intents behind these queries. The fundamental challenge in the interaction of these users and DBMS is that the users and DBMS represent intents in different forms.

Many such users may explore a database to find answers for various intents over a rather long period of time. For these users, database querying is an inherently interactive and continuous process. As both the user and DBMS have the same goal of the user receiving her desired information, the user and DBMS would like to gradually improve their understanding of each other and reach a *common language of representing intents* over the course of various queries and interactions. The user may learn more about the structure and content of the database and how to express intents as she submits queries and observes the returned results. Also, the DBMS may learn more about how the user expresses her intents by leveraging user feedback on the returned results. The user feedback may include clicking on the relevant answers [33], or the signals sent in touch-based devices [20]. Ideally, the user and DBMS should establish as quickly as possible this common representation of intents in which the DBMS accurately understands all or most user's queries.

Researchers have developed systems that leverage user feedback to help the DBMS understand the intent behind ill-specified and vague queries more precisely [6]. These systems, however, generally assume that a user does *not* modify her method of expressing intents throughout her interaction with the DBMS. For example, they maintain that the user picks queries to express an intent according to a fixed probability distribution. It is known that the learning methods that are useful in a static setting do not deliver desired outcomes in a setting where all agents may modify their strategies [14]. Hence, one may not be able to use current techniques to help the DBMS understand the users' information need in a rather long-term interaction.

To the best of our knowledge, the impact of user learning on database interaction has been generally ignored. In this paper, we propose a novel framework that formalizes the interaction between the user and the DBMS as a game with identical interest between two active and potentially rational agents: the user and DBMS. The common goal of the user and DBMS is to reach a mutual understanding on expressing information needs in the form of keyword queries. In each interaction, the user and DBMS receive certain payoffs according to how much the returned results are relevant

©ACM 2018. This is a minor revision of the paper entitled The Data Interaction Game published in SIGMOD'18, ISBN978-1-4503-4703-7, June 10 - June 15, 2018, New York, NY, USA. DOI: http://doi.acm.org/10.1145/3183713.3196899

to the intent behind the submitted query. The user receives her payoff by consuming the relevant information and the DBMS becomes aware of its payoff by observing the user's feedback on the returned results. We believe that such a game-theoretic framework naturally models the long-term interaction between the user and DBMS. We explore the user learning mechanisms and propose algorithms for the DBMS to improve its understanding of intents behind the user queries effectively and efficiently over large databases. In particular, we make the following contributions:

- We model the long term interaction between the user and DBMS using keyword queries as a particular type of game called a signaling game [9] in Section 2.
- Using extensive empirical studies over a real-world interaction log, we show that users modify the way they express their information need over their course of interactions in Section 3. We also show that this adaptation is accurately modeled by a well-known reinforcement learning algorithm [27] in experimental game-theory.
- We describe our data interaction system that provides an efficient implementation of our reinforcement learning method on large relational databases in Section 5. In particular, we first propose an algorithm that implements our learning method called *Reservoir*. Then, using certain mild assumptions and the ideas of sampling over relational operators, we propose another algorithm called *Poisson-Olken* that implements our reinforcement learning scheme and considerably improves the efficiency of *Reservoir*.
- We report the results of our empirical studies on measuring the effectiveness of our reinforcement learning method and the efficiency of our algorithms using real-world and large interaction workloads, queries, and databases in Section 6. Our results indicate that our proposed reinforcement learning method is more effective than the start-of-the-art algorithm for long-term interactions. They also show that *Poisson-Olken* can process queries over large databases faster than the *Reservoir* algorithm.

2. A GAME-THEORETIC FRAMEWORK

Users and DBMSs typically achieve a common understanding *gradually* and using a *querying/feedback* paradigm. After submitting each query, the user may revise her strategy of expressing intents based on the returned result. If the returned answers satisfy her intent to a large extent, she may keep using the same query to articulate her intent. Otherwise, she may revise her strategy and choose another query to express her intent in the hope that the new query will provide her with more relevant answers. We will describe this behavior of users in Section 3 in more detail. The user may also inform the database system about the degree by which the returned answers satisfy the intent behind the query using explicit or implicit feedback, e.g., click-through information [13]. The DBMS may update its interpretation of the query according to the user's feedback.

Intuitively, one may model this interaction as a game between two agents with identical interests in which the agents communicate via sharing queries, results, and feedback on the results. In each interaction, both agents will receive some reward according to the degree by which the returned result for a query matches its intent. The user receives her rewards in the form of answers relevant to her intent and the DBMS receives its reward through getting positive feedback on the returned results. The final goal of both agents is to maximize the amount of reward they receive during the course of their interaction.

2.1 Intent

An *intent* represents an information need sought after by the user. Current keyword query interfaces over relational databases generally assume that each intent is a query in a sufficiently expressive query language in the domain of interest, e.g., Select-Project-Join subset of SQL [8, 18]. Our framework and results are orthogonal to the language that precisely describes the users' intents. Table 1 illustrates a database with schema *Univ(Name, Abbreviation, State, Rank)* that contains information about university rankings. A user may want to find the information about university *MSU* in Michigan, which is precisely represented by the intent e_2 in Table 2(a), which using the Datalog syntax [1] is: $ans(z) \leftarrow Univ(x, `MSU', `MI', z)$.

2.2 Query

Users' articulations of their intents are *queries.* Many users do not know the formal query language, e.g., SQL, that precisely describes their intents. Thus, they may prefer to articulate their intents in languages that are easy-to-use, relatively less complex, and ambiguous such as keyword query language [18, 8]. In the proposed game-theoretic frameworks for database interaction, we assume that the user expresses her intents as keyword queries. More formally, we fix a countably infinite set of terms, i.e., keywords, T. A *keyword query* (query for short) is a nonempty (finite) set of terms in T. Consider the database instance in Table 1. Table 2 depicts a set of intents and queries over this database. Suppose the user wants to find the information about Michigan State University in Michigan, i.e. the intent e_2. Because the user does not know any formal database query language and may not be sufficiently familiar with the content of the data, she may express intent e_2 using q_2 : *'MSU'*.

Some users may know a formal database query language that is sufficiently expressive to represent their intents. Nevertheless, because they may not know precisely the content and schema of the database, their submitted queries may not always be the same as their intents [6]. For example, a user may know how to write a SQL query. But, since she may not know the state abbreviation *MI*, she may articulate intent e_2 as $ans(z) \leftarrow Univ(x, `MSU', y, z)$, which is different from e_2. We plan to extend our framework for these scenarios in future work. But, in this paper, we assume that users articulate their intents as keyword queries.

2.3 User Strategy

The user strategy indicates the likelihood that the user submits query q given that her intent is e. In practice, a user has finitely many intents and submits finitely many queries in a finite period of time. Hence, we assume that the sets of the user's intents and queries are finite. We index each user's intent and query by $1 \leq i \leq m$ and $1 \leq j \leq n$, respectively. A user strategy, denoted as U, is a $m \times n$ row-stochastic matrix from her intents to her queries. The matrix on the top of Table 3(a) depicts a user strategy using intents and queries in Table 2. According to this strategy, the user submits query q_2 to express intents e_1, e_2, and e_3.

Table 1: A database instance of relation Univ

Name	Abbreviation	State	Rank
Missouri State University	MSU	MO	20
Mississippi State University	MSU	MS	22
Murray State University	MSU	KY	14
Michigan State University	MSU	MI	18

Table 2: Intents and Queries

2(a) Intents

Intent#	Intent
e_1	$ans(z) \leftarrow Univ(x, \text{'}MSU\text{'}, \text{'}MS\text{'}, z)$
e_2	$ans(z) \leftarrow Univ(x, \text{'}MSU\text{'}, \text{'}MI\text{'}, z)$
e_3	$ans(z) \leftarrow Univ(x, \text{'}MSU\text{'}, \text{'}MO\text{'}, z)$

2(b) Queries

Query#	Query
q_1	'MSU MI'
q_2	'MSU'

Table 3: Two strategy profiles over the intents and queries in Table 2. User and DBMS strategies at the top and bottom, respectively.

3(a) A strategy profile

	q_1	q_2
e_1	0	1
e_2	0	1
e_3	0	1

	e_1	e_2	e_3
q_1	0	1	0
q_2	0	1	0

3(b) Another strategy profile

	q_1	q_2
e_1	0	1
e_2	1	0
e_3	0	1

	e_1	e_2	e_3
q_1	0	1	0
q_2	0.5	0	0.5

2.4 DBMS Strategy

The DBMS interprets queries to find the intents behind them. It usually interprets queries by mapping them to a subset of SQL [8, 16]. Since the final goal of users is to see the result of applying the interpretation(s) on the underlying database, the DBMS runs its interpretation(s) over the database and returns its results. Moreover, since the user may *not* know SQL, suggesting possible SQL queries may not be useful. A DBMS may *not* exactly know the language that can express all users' intents. Current usable query interfaces, including keyword query systems, select a query language for the interpreted intents that is sufficiently complex to express many users' intents and is simple enough so that the interpretation and running its outcome(s) are done efficiently [8].

To better leverage users feedback during the interaction, the DBMS must show the results of and get feedback on a sufficiently diverse set of interpretations [15, 31]. Of course, the DBMS should ensure that this set of interpretations is relatively relevant to the query, otherwise the user may become discouraged and give up querying. This dilemma is called the *exploitation versus exploration* trade-off. A DBMS that only *exploits*, returns top-ranked interpretations according to its scoring function. Hence, the DBMS may adopt a *stochastic strategy* to both exploit and explore: it randomly selects and shows the results of intents such that the ones with higher scores are chosen with larger probabilities [15, 31]. In this approach, users are mostly shown results of interpretations that are relevant to their intents according to the current knowledge of the DBMS and provide feedback on a relatively diverse set of interpretations. More formally, given Q is a set of all keyword queries, the DBMS strategy D is a stochastic mapping from Q to L, where L is some interpretation of the keyword query that contains some tuples from the underlying database. The matrix on the bottom of Table 3(a) depicts a DBMS strategy for the intents and queries in Table 2. Based on this strategy, the DBMS uses a exploitative strategy and always interprets query q_2 as e_2. The matrix on the bottom of Table 3(b) depicts another DBMS strategy for the same set of intents and queries. In this example, DBMS uses a randomized strategy and does both exploitation and exploration. For instance, it explores e_1 and e_3 to answer q_2 with equal probabilities, but it always returns e_2 in the response to q_1.

2.5 Interaction & Adaptation

The data interaction game is a repeated game with identical interest between two players, the user and the DBMS. At each round of the game, i.e., a single interaction, the user selects an intent according to the prior probability distribution π. She then picks the query q according to her strategy and submits it to the DBMS. The DBMS observes q and interprets q based on its strategy, and returns the results of the interpretation(s) on the underlying database to the user. The user provides some feedback on the returned tuples and informs the DBMS how relevant the tuples are to her intent. In this paper, we assume that the user informs the DBMS if some tuples satisfy the intent via some signal, e.g., selecting the tuple, in some interactions.

Next, we compute the expected payoff of the players. Since DBMS strategy D maps each query to a finite set of interpretations, and the set of submitted queries by a user, or a population of users, is finite, the set of interpretations for all queries submitted by a user, denoted as L^s, is finite. Hence, we show the DBMS strategy for a user as an $n \times o$ row-stochastic matrix from the set of the user's queries to the set of interpretations L^s. We index each interpretation in L^s by $1 \leq \ell \leq o$. Each pair of the user and the DBMS strategy, (U,D), is a *strategy profile.* The expected payoff for both players with strategy profile (U,D) is as follows, where $r(e_i, e_\ell)$ is some effectiveness metric such as *precision at k* [21].

$$u_r(U, D) = \sum_{i=1}^{m} \pi_i \sum_{j=1}^{n} U_{ij} \sum_{\ell=1}^{o} D_{j\ell}\ r(e_i, e_\ell), \tag{1}$$

The expected payoff reflects the degree by which the user and DBMS have reached a common language for communication. This value is high for the case in which the user knows which queries to pick to articulate her intents and the DBMS returns the results that satisfy the intents behind the user's queries. Hence, this function reflects the success of the communication and interaction. For example, given that all intents have equal prior probabilities, intuitively, the strategy profile in Table 3(b) shows a larger degree of mutual understanding between the players than the one in Table 3(a). This is reflected in their values of expected payoff as the expected payoffs of the former and latter are $\frac{2}{3}$ and $\frac{1}{3}$, respectively. We note that the DBMS may *not* know the set of users' queries beforehand and does *not* compute the expected payoff directly. Instead, it uses query answering algorithms that leverage user feedback, such that the expected payoff improves over the course of several interactions.

3. USER LEARNING MECHANISM

It is well established that humans show reinforcement behavior in learning [29, 24]. Many lab studies with human subjects conclude that one can model human learning using reinforcement learning models [29, 24]. The exact reinforcement learning method used by a person, however, may vary based on her capabilities and the task at hand. We have performed an empirical study of a real-world interaction log to find the reinforcement learning method(s) that best explain the mechanism by which users adapt their strategies during interaction with a DBMS.

3.1 Human Learning Schemes

To provide a comprehensive comparison, we evaluate six reinforcement learning methods used to model human learning in experimental game theory and/or Human Computer Interaction (HCI) [27, 5]. These methods mainly vary based on 1) the degree by which the user considers past interactions when computing future strategies, 2) how they update the user strategy, and 3) the rate by which they update the user strategy. *Win-Keep/Lose-Randomize* keeps a query with non-zero reward in past interactions for an intent. If such a query does not exist, it picks a query randomly. *Latest-Reward* reinforces the probability of using a query to express an intent based on the most recent reward of the query to convey the intent. *Bush and Mosteller's* and *Cross's* models increases (decreases) the probability of using a query based its past success (failures) of expressing an intent. A query is successful if it delivers a reward more than a given threshold, e.g., zero. *Roth and Erev's* model uses the aggregated reward from past interactions to compute the probability by which a query is used. *Roth and Erev's modified* model is similar to Roth and Erev's model, with an additional parameter that determines to what extent the user *forgets* the reward received for a query in past interactions.

3.2 Empirical Analysis

Interaction Logs: We use an anonymized Yahoo! interaction log for our empirical study, which consists of keyword queries submitted to a Yahoo! search engine in July 2010 [32]. We have used three different contiguous subsamples of this log whose information is shown in Table 4. The duration of each subsample is the time between the timestamp of the first and last interaction records. The records of the 8H-interaction sample appear at the beginning of the the 43H-interaction sample, which themselves appear at the beginning of the 101H-interaction sample.

Intent & Reward: Accompanying the interaction log is a set of *relevance judgment scores* for each query and result pair. Each relevance judgment score is a value between 0 and 4 and shows the degree of relevance of the result to the query, with 0 meaning not relevant at all and 4 meaning the most relevant result. We define the intent behind each query as the set of results with non-zero relevance scores. We use the standard ranking quality metric Normalized Discounted Cumulative Gain (NDCG) for the returned results of a query as the reward in each interaction as it models different levels of relevance [21]. The value of NDCG is between 0 and 1 and it is 1 for the most effective list.

Training & Testing: We have used a set of 5,000 records that appear in the interaction log immediately before the first subsample of Table 4 and found the optimal values for parameters using grid search and the sum of squared errors. We train and test a single user strategy over each subsample and model, which represents the strategy of the user population in each subsample. After estimating parameters, we train the user strategy using each model over 90% of the total number of records in each selected subsample in the order by which the records appear in the interaction log and test over the remaining 10% using the user strategy computed at the end of the training phase. We report the mean squared errors over all intents in the testing phase for each subsample and model in Table 5. A lower mean squared error implies that the model more accurately represents the users' learning method. We have excluded the Latest Reward results from the figure as they are an order of magnitude worse than the others.

Table 4: Subsamples of Yahoo! interaction log

Duration	#Interactions	#Users	#Queries	#Intents
˜8H	622	272	111	62
˜43H	12323	4056	341	151
˜101H	195468	79516	13976	4829

Table 5: Accuracies of learning over the subsamples of Table 4

Methods	Duration		
	101H	43H	8H
Bush and Mosteller's	0.0672	0.1880	0.2434
Cross's	0.0686	0.1908	0.2472
Roth and Erev's	**0.0666**	**0.1827**	0.2522
Roth and Erev's Modified	**0.0666**	**0.1827**	0.2522
Win-Keep/Lose-Randomize	0.0713	0.1876	**0.2364**

Results: Win-Keep/Lose-Randomize performs surprisingly more accurately than other methods for the 8H-interaction subsample. It indicates that in short-term and/or beginning of their interactions, users may not have enough interactions to leverage a more complex learning scheme and use a rather simple mechanism to update their strategies. Both Roth and Erev's methods use the accumulated reward values to adjust the user strategy gradually. Hence, they cannot precisely model user learning over a rather short interaction and are less accurate than relatively more aggressive learning models such as Bush and Mosteller's and Cross's over this subsample. Both Roth and Erev's deliver the same result and outperform other methods in the 43-H and 101-H subsamples. Win-Keep/Lose-Randomize is the least accurate method over the two larger subsamples. Since larger subsamples provide more training data, the predication accuracy of all models improves as the interaction subsamples becomes larger. The learned value for the *forget* parameter in the Roth and Erev's modified model is very small and close to zero in our experiments, therefore, it generally acts like the Roth and Erev's model.

Long-term communications between users and DBMS may include multiple sessions. Since Yahoo! query workload contains the time stamps and user ids of each interaction, we have been able to extract the starting and ending times of each session. Our results indicate that as long as the user and DBMS communicate over sufficiently many of interactions, e.g., about 10k for Yahoo! query workload, the users follow the Roth and Erev's model of learning. Given that the communication of the user and DBMS involve sufficiently many interactions, we have *not* observed any difference in the mechanism by which users learn based on the numbers of sessions in the user and DBMS communication.

Conclusion: Our analysis indicates that users show a substantially intelligent behavior when adopting and modifying their strategies over relatively medium and long-term interactions. They leverage their past interactions and their outcomes, i.e., have an effective long-term memory. This behavior is most accurately modeled using Roth and Erev's model. Hence, in the rest of the paper, we set the user learning method to this model.

4. LEARNING ALGORITHM FOR DBMS

Current systems generally assume that a user does *not* learn and/or modify her method of expressing intents throughout her interaction with the DBMS. However, it is known that the learning methods that are useful in static settings do not deliver desired outcomes in the dynamic ones [3]. Moreover, it has been shown that if the players do *not* use the right learning algorithms in games with identical interests, the game and its payoff may not converge to any desired states [28]. Thus, choosing the correct learning mechanism for the DBMS is crucial to improve the payoff and converge to a desired state.

4.1 DBMS Reinforcement Learning

We adopt Roth and Erev's learning method for adaptation of the DBMS strategy, with a slight modification. The original Roth and Erev method considers only a single action space. In our work, this would translate to having only a single query. Instead we extend this such that each query has its own action space or set of possible intents. The adaptation happens over discrete time $t = 0, 1, 2, 3, \ldots$ instances where t denotes the tth interaction of the user and the DBMS. We refer to t simply as the iteration of the learning rule. For simplicity of notation, we refer to intent e_i and result s_ℓ as intent i and ℓ, respectively, in the rest of the paper. Hence, we may rewrite the expected payoff for both user and DBMS as:

$$u_r(U, D) = \sum_{i=1}^{m} \pi_i \sum_{j=1}^{n} U_{ij} \sum_{\ell=1}^{o} D_{j\ell} r_{i\ell},$$

where $r : [m] \times [o] \to \mathbb{R}^+$ is the effectiveness measure between the intent i and the result, i.e., decoded intent ℓ. With this, the reinforcement learning mechanism for the DBMS adaptation is as follows.

a. Let $R(0) > 0$ be an $n \times o$ initial reward matrix whose entries are strictly positive.

b. Let $D(0)$ be the initial DBMS strategy with $D_{j\ell}(0) = \frac{R_{j\ell}(0)}{\sum_{\ell=1}^{o} R_{j\ell}(0)} > 0$ for all $j \in [n]$ and $\ell \in [o]$.

c. For iterations $t = 1, 2, \ldots$, do

    i. If the user's query at time t is $q(t)$, DBMS returns a result $E(t) \in E$ with probability:

    $$P(E(t) = i' \mid q(t)) = D_{q(t)i'}(t).$$

    ii. User gives a reward $r_{ii'}$ given that i is the intent of the user at time t. Note that the reward depends both on the intent i at time t and the result i'. Then, set

    $$R_{j\ell}(t+1) = \begin{cases} R_{j\ell}(t) + r_{i\ell} & \text{if } j = q(t) \text{ and } \ell = i' \\ R_{j\ell}(t) & \text{otherwise} \end{cases}. \tag{2}$$

    iii. Update the DBMS strategy by

    $$D_{ji}(t+1) = \frac{R_{ji}(t+1)}{\sum_{\ell=1}^{o} R_{j\ell}(t+1)}, \tag{3}$$

    for all $j \in [n]$ and $i \in [o]$.

In the above algorithm $R(t)$ is simply the reward matrix at time t. We have also proved the following:

THEOREM 4.1. *The proposed learning algorithm in Section 4.1 converges almost surely when the user learns using Roth and Erev's model.*

The above result implies that the effectiveness of the DBMS, stochastically speaking, increases as time progresses when the learning rule in Section 4.1 is utilized. The user may also learn at a relatively slow rate such that from the perspective of the database it seems as though the user isn't learning. Of course, the user may not perform any learning. Our results also hold for the case when the user doesn't learn. We have also proved that the payoff of the two agents only increases or remains the same. To see all proofs in full, we refer the reader to our published work [22].

It is quite costly to materialize and maintain the strategy of the DBMS as shown in the previous examples. Thus, we maintain the strategy and reinforcements in a constructed feature space using n-grams for each attribute value. Our mapping is then a mapping from query features to tuple features.

5. EFFICIENT QUERY ANSWERING OVER RELATIONAL DATABASES

An efficient implementation of our algorithm proposed in Section 4 over large relational databases poses two challenges. First, since the set of possible interpretations and their results for a given query is enormous, one has to find efficient ways of maintaining users' reinforcements and updating DBMS strategy. Second, keyword and other usable query interfaces over databases normally return the top-k tuples according to some scoring functions [16, 8]. Due to a series of seminal works by database researchers [12], there are efficient algorithms to find such a list of answers. Nevertheless, our reinforcement learning algorithm uses a randomized semantics for answering algorithms in which candidate tuples are associated a probability for each query that reflects the likelihood by which it satisfies the intent behind the query. The tuples must be returned randomly according to their associated probabilities. Using (weighted) sampling to answer SQL queries with aggregation functions approximately and efficiently is an active research area [17]. However, there has not been any attempt on using a randomized strategy to answer so-called point queries over relational data and achieve a balanced exploitation-exploration trade-off efficiently.

5.1 Keyword Query Interface

We use the current architecture of keyword query interfaces over relational databases that directly use schema information to interpret the input keyword query [8]. A notable example of such systems is IR-Style [16]. We provide an overview of the basic concepts of such a system. We refer the reader to [16, 8] for more explanation.

Tuple-set: Given keyword query q, a *tuple-set* is a set of tuples in a base relation that contain some terms in q. After receiving q, the query interface uses an inverted index to compute a set of tuple-sets. For instance, consider a database of products with relations *Product(pid, name)*, *Customer(cid, name)*, and *ProductCustomer(pid, cid)* where *pid* and *cid* are numeric strings. Given query *iMac John*, the query interface returns a tuple-set from *Product* and a tuple-set from *Customer* that match at least one term in the query.

Candidate Network: A *candidate network* is a join expression that connects the tuple-sets via primary key-foreign key relationships. A candidate network joins the tuples in different tuple-sets and produces joint tuples that contain the terms in the input keyword query. One may consider the candidate network as a join tree expression whose leaves are tuple-sets. For instance, one candidate network for the aforementioned database of products is *Product* $\bowtie$ *ProductCustomer* $\bowtie$ *Customer*. To connect tuple-sets via primary key-foreign key links, a candidate network may include base relations whose tuples may not contain any term in the query, e.g., *ProductCustomer* in the preceding example. Given a set of tuple-sets, the query interface uses the schema of the database and progressively generates candidate networks that can join the tuple-sets. For efficiency considerations, keyword query interfaces limit the number of relations in a candidate network to be lower than a given threshold. Keyword query interfaces normally compute the score of joint tuples by summing up the scores of their constructing tuples multiplied by the inverse of the number of relations in the candidate network to penalize long joins [8]. We use the same scoring scheme. We also consider each (joint) tuple to be candidate answer to the query if it contains at least one term in the query.

5.2 Efficient Exploitation & Exploration

We propose the following two algorithms to generate a weighted random sample of size k over all candidate tuples for a query.

5.2.1 *Reservoir*

To provide a random sample, one may calculate the total scores of all candidate answers to compute their sampling probabilities. Because this value is not known beforehand, one may use weighted reservoir sampling [7] to deliver a random sample without knowing the total score of candidate answers in a single scan of the data as follows. *Reservoir* occurs after the complete joins of the candidate network have be computed. Thus, it samples over tuples in the individual tables and tuples in the joined tables. Reservoir generates the list of answers only after computing the results of all candidate networks, therefore, users have to wait for a long time to see any result. It also computes the results of all candidate networks by performing their joins fully, which may be inefficient. We propose the following optimizations to improve its efficiency and reduce the users' waiting time.

5.2.2 *Poisson-Olken*

Poisson-Olken algorithm uses Poisson sampling to output progressively the selected tuples as it processes each candidate network [25]. First, when a join needs to be constructed between multiple tables in a candidate network, tuples are only joined based on some statistics collected prior to interaction. These include how likely a given tuple might join with another and how many tuples are in each relation. As tuples are joined, they are sampled immediately, allowing the algorithm to return before it has performed the entire join.

The expected value of produced tuples in the *Poisson-Olken* algorithm is close to k. However, as opposed to reservoir sampling, there is a non-zero probability that *Poisson-Olken* may deliver fewer than k tuples. To drastically reduce this chance, one may use a larger value for k in the algorithm and reject the appropriate number of the resulting tuples after the algorithm terminates [7]. The resulting algorithm will not progressively produce the sampled tuples, but, as our empirical study in Section 6 indicates, it is faster than *Reservoir* over large databases with relatively many candidate networks as it does not perform any full join. For more details, including the algorithms, see our full publication in [22].

6. EMPIRICAL STUDY

6.1 Effectiveness

It is difficult to evaluate the effectiveness of online and reinforcement learning algorithms for information systems in a live setting with real users because it requires a very long time and a large amount of resources [31, 15, 26, 14]. Thus, most studies in this area use purely simulated user interactions [26, 15]. A notable exception is [31], which uses a real-world interaction log to simulate a live interaction setting. We follow a similar approach and use Yahoo! interaction log [32] to simulate interactions using real-world queries and dataset.

Strategy Initialization: We train a user strategy over the Yahoo! 43H-interaction log whose details are in Section 3 using Roth and Erev's method, which is deemed the most accurate to model user learning according to the results of Section 3. This strategy has 341 queries and 151 intents. The DBMS starts the interaction with an empty strategy and adds queries as it receives them, initialized with equal probabilities.

Algorithms: We compare the algorithm introduced in Section 4.1 against the state-of-the-art and popular algorithm for online learning in information retrieval called UCB-1 [26, 23]. It has been shown to outperform its competitors in several studies [23, 26]. It calculates a score for an intent e given the tth submission of query q as: $Score_t(q,e) = \frac{W_{q,e,t}}{X_{q,e,t}} + \alpha\sqrt{\frac{2ln\ t}{X_{q,e,t}}}$, in which X is how many times an intent was shown to the user, W is how many times the user selects a returned intent, and α is the exploration rate set between $[0,1]$. The first term in the formula prefers the intents that have received relatively more positive feedback, i.e., exploitation, and the second term gives higher scores to the intents that have been shown to the user less often and/or have *not* been tried for a relatively long time, i.e., exploration. UCB-1 assumes that users follow a fixed probabilistic strategy. Thus, its goal is to find the fixed but unknown expectation of the relevance of an intent to the input query, which is roughly the first term in the formula; by minimizing the number of unsuccessful trials.

Results: We simulate the interaction of a user population that starts with our trained user strategy with UCB-1 and our algorithm. We measure the effectiveness of the algorithms using the standard metric of Reciprocal Rank (RR) [21]. In each interaction, an intent is randomly picked from the set of intents in the user strategy by its prior probability and submitted to UCB-1 and our method. Afterwards, each algorithm returns a list of 10 answers and the user clicks on the top-ranked answer that is relevant to the query according to the relevance judgment information. We run our simulations for one million interactions.

Figure 1 shows the accumulated Mean Reciprocal Rank (MRR) over all queries in the simulated interactions. Our method delivers a higher MRR than UCB-1 and its MRR keeps improving over the duration of the interaction. UCB-1, however, increases the MRR at a much slower rate. Since UCB-1 is developed for the case where users do *not* change their strategies, it learns and commits to a fixed probabilistic mapping of queries to intents quite early in the interaction. We have also observed that our method allows users to try more varieties of queries to express an intent and learn the one(s) that convey the intent effectively. As UCB-1 commits to a certain mapping of a query to an intent early in the interaction, it may *not* return sufficiently many relevant answers if the user tries this query to express another intent. This new mapping, however, could be promising in the long-run. Hence, the user and UCB-1 strategies may stabilize in less than desirable states. Since our method does *not* commit to a fixed strategy that early, users may try this query for another intent and reinforce the mapping if they get relevant answers. Thus, users have more chances to try and pick a query for an intent that will be learned and mapped effectively to the intent by the DBMS.

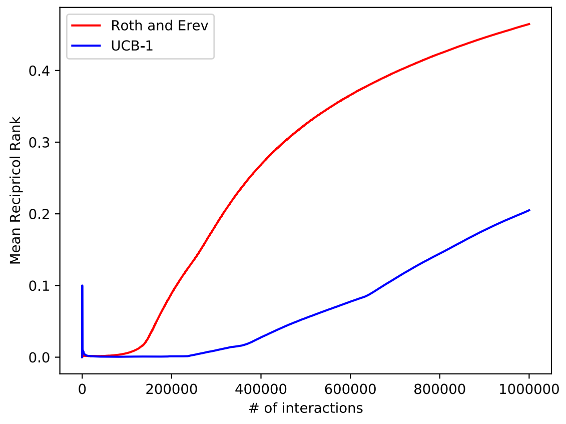


Figure 1: Mean reciprocal rank for 1,000,000 interactions

6.2 Efficiency

Databases and Queries: We have built two databases from Freebase (*developers.google.com/freebase*), *TV-Program* and *Play*. *TV-Program* contains 7 tables and consists of 291,026 tuples. *Play* contains 3 tables and consists of 8,685 tuples. For our queries, we have used two samples of 621 (459 unique) and 221 (141 unique) queries from Bing (*bing.com*) query log whose relevant answers after filtering our noisy clicks, are in *TV-program* and *Play* databases, respectively [10]. After submitting each query and getting some results, we simulate user feedback using the relevance information in the Bing log.

Query Processing: We have used Whoosh inverted index (*whoosh.readthedocs.io*) to index each table in databases. Whoosh recognizes the concept of table with multiple attributes, but cannot perform joins between different tables. Because the *Poisson-Olken* algorithm needs indexes over primary and foreign keys used to build candidate network, we have built hash indexes over these tables in Whoosh. Given an index-key, these indexes return the tuple(s) that match these keys inside Whoosh. To provide a fair comparison between *Reservoir* and *Poisson-Olken*, we have used these indexes to perform joins for both methods. We have limited the size of each candidate network to 5. Our system returns 10 tuples in each interaction for both methods.

Results: Table 6 depicts the time for processing candidate networks and reporting the results for both *Reservoir* and *Poisson-Olken* over *TV-Program* and *Play* databases over 1000 interactions. These results also show that *Poisson-Olken* is able to significantly improve the time for executing the joins in the candidate network, shown as performing joins in the table, over *Reservoir* in both databases. The improvement is more significant for the larger database, *TV-Program*. *Poisson-Olken* progressively produces tuples to show to user. But, we are not able to use this feature for all interactions. For a considerable number of interactions, *Poisson-Olken* does not produce 10 tuples, as explained in Section 5.2. Hence, we have to use a larger value of k and wait for the algorithm to finish in order to find a randomize sample of the answers as explained at the end of Section 5.2. Both methods have spent a negligible amount of time to reinforce the features, which indicate that using a rich set of features one can perform and manage reinforcement efficiently.

Table 6: Average candidate networks processing times in seconds for 1000 interactions

Database	Reservoir	Poisson-Olken
Play	0.078	0.042
TV Program	0.298	0.171

7. RELATED WORK

Database community has proposed several systems that help the DBMS learn the user's information need by showing examples to the user and collecting her feedback [19, 11, 4, 30, 2]. In these systems, a user *explicitly teaches* the system by labeling a set of examples potentially in several steps without getting any answer to her information need. Thus, the system is broken into two steps: first it learns the information need of the user by soliciting labels on certain examples from the user and then once the learning has completed, it suggests a query that may express the user's information need. These systems usually leverage active learning methods to learn the user intent by showing the fewest possible examples to the user [11]. However, ideally one would like to have a query interface in which the DBMS learns about the user's intents while answering her (vague) queries as our system does. As opposed to active learning methods, one should combine and balance exploration and learning with the normal query answering to build such a system. Moreover, current query learning systems assume that users follow a fixed strategy for expressing their intents. Also, we focus on the problems that arise in the long-term interaction that contain more than a single query and intent.

8. CONCLUSION

Many users do *not* know how to express their information needs. We showed that users learn and modify how they express their information needs during their interaction with the DBMS and modeled the interaction between the user and the DBMS as a game, where the players would like to establish a common mapping from information needs to queries via learning. As current query interfaces do *not* effectively learn the information needs behind queries in such a setting, we proposed a reinforcement learning algorithm for the DBMS that learns the querying strategy of the user effectively. We provided efficient implementations of this learning mechanisms over large databases.

Currently the algorithm proposed in this work does not consider optimal strategy profiles. In the future we would like to have the DBMS algorithm target these optimal strategy profiles such that the mutual understanding between the two players is optimal. Another question to ask is whether there are information preserving transformations of data, that can deliver a more effective interaction, e.g. merging some entities. Given that the aforementioned transformations are costly and they do improve the interaction, we would need to find the most cost-effective ones.

9. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases: The Logical Level.* Addison-Wesley, 1994.

[2] A. Abouzied, D. Angluin, C. H. Papadimitriou, J. M. Hellerstein, and A. Silberschatz. Learning and verifying quantified boolean queries by example. In *PODS*, 2013.

[3] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM journal on computing*, 32(1):48–77, 2002.

[4] A. Bonifati, R. Ciucanu, and S. Staworko. Learning join queries from user examples. *TODS*, 40(4), 2015.

[5] Y. Cen, L. Gan, and C. Bai. Reinforcement learning in information searching. *Information Research: An International Electronic Journal*, 18(1), 2013.

[6] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic information retrieval approach for ranking of database query results. *TODS*, 31(3), 2006.

[7] S. Chaudhuri, R. Motwani, and V. Narasayya. On random sampling over joins. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, SIGMOD '99, pages 263–274, New York, NY, USA, 1999. ACM.

[8] Y. Chen, W. Wang, Z. Liu, and X. Lin. Keyword search on structured and semi-structured data. In *SIGMOD*, 2009.

[9] I. Cho and D. Kreps. Signaling games and stable equilibria. *Quarterly Journal of Economics*, 102, 1987.

[10] E. Demidova, X. Zhou, I. Oelze, and W. Nejdl. Evaluating Evidences for Keyword Query Disambiguation in Entity Centric Database Search. In *DEXA*, 2010.

[11] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. Explore-by-example: An automatic query steering framework for interactive data exploration. In *SIGMOD*, 2014.

[12] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '01, pages 102–113, New York, NY, USA, 2001. ACM.

[13] L. A. Granka, T. Joachims, and G. Gay. Eye-tracking analysis of user behavior in www search. In *SIGIR*, 2004.

[14] A. Grotov and M. de Rijke. Online learning to rank for information retrieval: Sigir 2016 tutorial. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, pages 1215–1218, New York, NY, USA, 2016. ACM.

[15] K. Hofmann, S. Whiteson, and M. de Rijke. Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval. *Information Retrieval*, 16(1):63–90, 2013.

[16] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-Style Keyword Search over Relational Databases. In *VLDB 2003*.

[17] S. Idreos, O. Papaemmanouil, and S. Chaudhuri. Overview of data exploration techniques. In *SIGMOD*, 2015.

[18] H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu. Making database systems usable. In *SIGMOD*, 2007.

[19] H. Li, C.-Y. Chan, and D. Maier. Query from examples: An iterative, data-driven approach to query construction. *PVLDB*, 8(13), 2015.

[20] E. Liarou and S. Idreos. dbtouch in action database kernels for touch-based data exploration. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 1262–1265, 2014.

[21] C. Manning, P. Raghavan, and H. Schutze. *An Introduction to Information Retrieval.* Cambridge University Press, 2008.

[22] B. McCamish, V. Ghadakchi, A. Termehchy, B. Touri, and L. Huang. The data interaction game. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 83–98, New York, NY, USA, 2018. ACM.

[23] T. Moon, W. Chu, L. Li, Z. Zheng, and Y. Chang. An online learning framework for refining recency search results with user click feedback. *ACM Transactions on Information Systems (TOIS)*, 30(4):20, 2012.

[24] Y. Niv. The neuroscience of reinforcement learning. In *ICML*, 2009.

[25] F. Olken. *Random Sampling from Databases.* PhD thesis, University of California, Berkeley, 1993.

[26] F. Radlinski, R. Kleinberg, and T. Joachims. Learning diverse rankings with multi-armed bandits. In *Proceedings of the 25th international conference on Machine learning*, pages 784–791. ACM, 2008.

[27] A. E. Roth and I. Erev. Learning in extensive-form games: Experimental data and simple dynamic models in the intermediate term. *Games and economic behavior*, 8(1):164–212, 1995.

[28] L. Shapley. Some topics in two-person games. *Advances in game theory*, 52:1–29, 1964.

[29] H. Shteingart and Y. Loewenstein. Reinforcement learning and human behavior. *Current Opinion in Neurobiology*, 25:93–98, 04/2014 2014.

[30] Q. Tran, C. Chan, and S. Parthasarathy. Query by output. In *SIGMOD*, 2009.

[31] A. Vorobev, D. Lefortier, G. Gusev, and P. Serdyukov. Gathering additional feedback on search results by multi-armed bandits with respect to production ranking. In *WWW*, pages 1177–1187. International World Wide Web Conferences Steering Committee, 2015.

[32] Yahoo! Yahoo! webscope dataset anonymized Yahoo! search logs with relevance judgments version 1.0. labs.yahoo.com/Academic_Relations, 2011. [Online; accessed 5-January-2017].

[33] Y. Yue, J. Broder, R. Kleinberg, and T. Joachims. The k-armed dueling bandits problem. *J. Comput. Syst. Sci.*, 78(5), 2012.

Technical Perspective for: MATLANG: Matrix Operations and Their Expressive Power

Dan Suciu
University of Washington

The main processing paradigm in data management is bulk processing. As introduced by Codd in the early 70's, under this paradigm relations are processed in bulk, one operator at a time. When applied to relations, this paradigm leads to relational algebra, and its variants, relational calculus, and SQL. Over the years, data management was faced with the challenge of extending bulk processing operators to new kinds of data, and/or new kinds of queries: nested relations, semistructured data, recursive queries. Each such extension requires significant systems development, which should be accompanied, in fact preceded, by a careful study of the expressive power of the new language. Is it as expressive, more expressive, or less expressive than relational algebra? The answer to this question has profound implications on the ability of data processing engines to optimize, compute, distribute, reuse queries in that language. For example, extending relational algebra with nested relations does not increase its expressive power, while extending it with fixpoint does, explaining why modern query engines have an easier time supporting JSON than recursion.

Today, data management is faced with a new task: incorporate into the relational engine linear algebra operations required by machine learning algorithms. A superficial inspection suggests that this is easy, because operations such as matrix multiplication, transpose, addition, are already expressible in SQL. In fact, in SciDB users indeed express these operations in SQL, and the engine can already apply the same optimizations as for general SQL queries. But this answer is unsatisfactory. Most popular ML or linear algebra systems today, such as NumPy, ScaLAPAK, SystemML, support *only* linear algebra operations, and, because of that, they outperform relational engines for the tasks they are designed to do (see reference [39] in the paper). This raises a natural and important question: what is the expressive power of linear algebra, and how does it relate to that of the relational algebra?

The paper by Brijder, Geerts, Van den Bussche and Weerwag, first published in ICDT'2018, initiates the study into precisely this question. This is undoubtedly only the first paper in this line, meant mostly to raise the question; more are likely to follow, for example see references [15] and [8], and the important results there.

The first hurdle the paper faces is that there is no standard linear algebra language. The authors propose one such language, MATLANG, which includes the usual operations on matrices, and not much else. The second hurdle is that expressions in linear algebra and in relational algebra have different types, hence are not directly comparable. To circumvent that, the authors focus on the graph properties that these languages can express, since graphs can be represented both as relations and as matrices. With these assumptions, the paper establishes a few results comparing the two languages. For example, properties in linear algebra can be expressed in FO^3 (which can be thought of as the restriction of relational algebra to intermediate results of arity at most 3), however, if one adds a matrix inverse operation to linear algebra then one can express graph connectivity, suggesting that this language is closer to the extension of relational algebra with fixpoints. The reader will enjoy the arguments used to prove these statements, and will definitely be enticed to follow future progress on the important study of the expressive power of linear algebra.

MATLANG: Matrix operations and their expressive power

Robert Brijder
Hasselt University
Hasselt, Belgium
robert.brijder@uhasselt.be

Floris Geerts
University of Antwerp
Antwerp, Belgium
floris.geerts@uantwerpen.be

Jan Van den Bussche
Hasselt University
Hasselt, Belgium
jan.vandenbussche@uhasselt.be

Timmy Weerwag
Hasselt University
Hasselt, Belgium

ABSTRACT

We investigate the expressive power of MATLANG, a formal language for matrix manipulation based on common matrix operations and linear algebra. The language can be extended with the operation inv for inverting a matrix. In MATLANG + inv we can compute the transitive closure of directed graphs, whereas we show that this is not possible without inversion. Indeed we show that the basic language can be simulated in the relational algebra with arithmetic operations, grouping, and summation. We also consider an operation eigen for diagonalizing a matrix. It is defined such that for each eigenvalue a set of orthogonal eigenvectors is returned that span the eigenspace of that eigenvalue. We show that inv can be expressed in MATLANG + eigen. We put forward the open question whether there are boolean queries about matrices, or generic queries about graphs, expressible in MATLANG + eigen but not in MATLANG + inv. Finally, the evaluation problem for MATLANG + eigen is shown to be complete for the complexity class $\exists\mathbf{R}$.

1. INTRODUCTION

In view of the importance of large-scale statistical and machine learning (ML) algorithms in the overall data analytics workflow, database systems are in the process of being redesigned and extended to allow for a seamless integration of ML algorithms and mathematical and statistical frameworks, such as R, SAS, and MATLAB, with existing data manipulation and data querying functionality [42, 19, 5, 38, 10, 27, 21]. In particular, data scientists often use *matrices* to represent their data, as opposed to using the relational data model, and create custom data analytics algorithms using *linear algebra*, instead of writing SQL queries. Here, linear algebra algorithms are expressed in a declarative manner by composing basic linear algebra constructs such as matrix multiplication, matrix transposition, element-wise operations on the entries of matrices, solving nonsingular systems of linear equations (matrix inversion), diagonalization (eigenvalues and eigenvectors), singular value decomposition, just to name a few. The main challenges from a database system's perspective are to ensure scalability by providing physical data independence and optimizations. We refer to [39] for an overview of the different systems addressing these challenges.

In this context, the following natural questions arise: Which linear algebra constructs need to be supported to perform certain data analytical tasks? Does the additional support for certain linear algebra operations increase the overall functionality? Can a linear algebra algorithm be rewritten, in an equivalent way, to an algorithm using a smaller number of linear algebra operations? Such questions have been extensively studied for "classical" query languages (fragments and extensions of SQL) in database theory and finite model theory [1, 26]. Indeed, the questions raised all relate to the *expressive power* of query languages. In this paper we enroll in the investigation of the expressive power of *matrix query languages*.

As a starting point we focus on matrices and matrix query languages alone, leaving the study of the expressive power of languages that operate on *both* relational data and matrices for future work. Even this "matrix only" setting turns out to be quite interesting and challenging on its own.

To set the stage, we need to formally define what we mean by a matrix query language. There has been work in finite model theory and logic to understand the capability of certain logics to express linear algebra operations [13, 12, 20]. In particular, the extent to which fixpoint logics with counting and their extension with so-called rank operators can express linear algebra has been considered. The motivation for that line of work is mainly to find a logical characterization of polynomial-time computability and less so in understanding the expressive power of specific linear algebra operations.

In this paper, we take the opposite approach in which we define a basic matrix query language, referred to as MATLANG, which is built up from *basic* linear algebra operations, supported by linear algebra systems such as R and MATLAB, and then closing these operations under *composition*. All basic linear algebra operations supported in MATLANG stem from "atomic" operations supported in these popular linear algebra packages. While many other operations are supported by these packages, we feel that they are somewhat less atomic. We present examples later on, showing that MATLANG is indeed capable of expressing common matrix manipulations. In fact, we propose MATLANG as *an analog for matrices of* *the relational algebra for relations.*

To study the expressive power of MATLANG, we relate it to the relational algebra with aggregates [25, 30]. In fact, it turns out that MATLANG is already subsumed by aggregate logic with only *three* nonnumerical variables. Conversely, MATLANG can express all queries from graph databases (binary relational structures) to binary relations that can be expressed in first-order logic with three variables. In contrast, the four-variable query asking if the graph contains a *four-clique*, is not expressible. We note that the connection with three-variable logics has recently been strengthened [15].

We thus see that, for example, when data analysts want to check for four-cliques in a graph, more advanced linear algebra operations than those in MATLANG need to be considered. Similarly, extracting information related to the connectivity of graphs requires extending MATLANG. We consider two such extensions in the paper: extending MATLANG with matrix inversion (inv) and extending MATLANG with an operation (eigen) to compute eigenvectors. Since no unique set of eigenvectors exists, the eigen operation is intrinsically *non-deterministic.*

We show that MATLANG + inv is strictly more expressive than MATLANG. Indeed, the transitive closure of binary relations becomes expressible. The possibility of reducing transitive closure to matrix inversion has been pointed out by several researchers [29, 11, 35].

We show that MATLANG + eigen can express inversion by using a deterministic MATLANG + eigen expression (i.e., despite it using eigen, it always deterministically returns the inverse of a matrix, if it exists). The argument is well known from linear algebra, but our result shows that starting from the eigenvectors, MATLANG is expressive enough to construct the inverse.

We subsequently show that the *equivalence* of MATLANG + eigen expressions is decidable. Related to this is the question whether the *evaluation* of expressions in MATLANG + eigen is effectively computable. This may seem like an odd question, since linear algebra computations are done in practice. These evaluation algorithms, however, often use techniques from numerical mathematics [17], resulting in approximations of the precise result — here, we are interested in the exact result. In particular, we show that the input-output relation of an expression e in MATLANG + eigen, applied to input matrices of given dimensions, is definable in the *existential theory of the real numbers* (which is decidable [3, 4]), by a formula of size polynomial in the size of e and the given dimensions.

We finally show that, conversely, there exists a fixed expression (data complexity) in MATLANG + eigen for which the evaluation problem is $\exists\mathbf{R}$-complete, where $\exists\mathbf{R}$ is the class of problems that can be reduced in polynomial time to the existential theory of the reals [36, 37], even when restricted to input matrices with integer entries.

1.1 Related work

Programming languages to manipulate matrices trace back to the APL language [22]. Providing database support for matrices and multidimensional arrays has been a long-standing research topic [33], originally geared towards applications in scientific data management.

In [27], Lara is proposed as a domain-specific programming language written in Scala that provides both linear algebra (LA) and relational algebra (RA) constructs. This approach is taken one step further in [21] where it is shown that the RA operations and a number of LA operations can be defined in terms of three core operations called Ext, Union, and Join.

Another relevant related work is the FAQ framework [2], which focuses on the project-join fragment of the algebra for K-relations [18]. The connection between MATLANG and the algebra for K-relations is more deeply investigated in [8]. Yet another related formalism is that of logics with rank operators [13, 12, 32]. These operators solve $0,1$-matrices over finite fields, and increase the expressive power of established logics over abstract structures. In contrast, in this paper we are interested in queries on arbitrary matrices.

Modest changes to SQL in order to perform LA operations in a scalable way within relational databases are proposed in [31]. In this way, various linear algebra operations are implemented in an efficient way using the relational algebra.

While the previous work is focused on showing that relational algebra (appropriately extended) can serve as a platform for supporting large scale linear algebra operations, the focus of our work here is complementary. Indeed, we want to understand the precise expressive power of common linear algebra operations, as adequately formalized in the language MATLANG and its extensions (see [7] for more details).

2. MATLANG

2.1 Syntax and semantics

We assume a sufficient supply of *matrix variables*, which serve to indicate the inputs to expressions in MATLANG. The syntax of MATLANG expressions is defined by the grammar:

$e ::= M$	(matrix variable)
$\mid$ let $M = e_1$ in e_2	(local binding)
$\mid$ e^*	(conjugate transpose)
$\mid$ $\mathbf{1}(e)$	(one-vector)
$\mid$ $\mathsf{diag}(e)$	(diagonalization of a vector)
$\mid$ $e_1 \cdot e_2$	(matrix multiplication)
$\mid$ $\mathsf{apply}[f](e_1, \dots, e_n)$	(pointwise application, $f \in \Omega$)

In the last rule, f is the name of a function $f : \mathbf{C}^n \to \mathbf{C}$, where $\mathbf{C}$ denotes the complex numbers. Formally, the syntax of MATLANG is parameterized by a repertoire Ω of such functions, but for simplicity we will not reflect this in the notation. We will see various examples of MATLANG expressions below.

To define the semantics of MATLANG, we first define the basic matrix operations. Following practical matrix sublanguages such as those of R or MATLAB, we will work throughout with matrices over the complex numbers. However, a real-number version of the language could be defined as well.

Transpose: If A is a matrix then A^* is its conjugate transpose. So, if A is an $m \times n$ matrix then A^* is an $n \times m$ matrix and the entry $A^*_{i,j}$ is the complex conjugate of the entry $A_{j,i}$.

One-vector: If A is an $m \times n$ matrix then $\mathbf{1}(A)$ is the $m \times 1$ column vector consisting of all ones.

$$\mathbf{1}\begin{pmatrix}2 & \sqrt{3} & 4\\ 4 & 5 & 6\end{pmatrix} = \begin{pmatrix}1\\1\end{pmatrix} \qquad \mathsf{diag}\begin{pmatrix}6\\7\end{pmatrix} = \begin{pmatrix}6 & 0\\ 0 & 7\end{pmatrix}$$

$$\mathsf{apply}[\dot{-}](\begin{pmatrix}1 & 1 & 1\\ 0 & 1 & 1\\ 0 & 0 & 0\end{pmatrix}, \begin{pmatrix}0 & 0 & 1\\ 0 & 1 & 0\\ 1 & 0 & 1\end{pmatrix}) = \begin{pmatrix}1 & 1 & 0\\ 0 & 0 & 1\\ 0 & 0 & 0\end{pmatrix}$$

Figure 1: Some basic matrix operations of MATLANG.

Diag: If v is an $m \times 1$ column vector then $\mathsf{diag}(v)$ is the $m \times m$ diagonal square matrix with v on the diagonal and zero everywhere else.

Matrix multiplication: If A is an $m \times n$ matrix and B is an $n \times p$ matrix then the well known matrix multiplication AB is defined to be the $m \times p$ matrix where $(AB)_{i,j} = \sum_{k=1}^{n} A_{i,k}B_{k,j}$. In MATLANG we explicitly denote this as $A \cdot B$.

Pointwise application: If $A^{(1)}, \dots, A^{(n)}$ are matrices of the same dimensions $m \times p$, then $\mathsf{apply}[f](A^{(1)}, \dots, A^{(n)})$ is the $m \times p$ matrix C where $C_{i,j} = f(A^{(1)}_{i,j}, \dots, A^{(n)}_{i,j})$.

EXAMPLE 2.1. *The operations* $\mathbf{1}(A)$, $\mathsf{diag}(v)$, *and* $\mathsf{apply}[f](A^{(1)}, \dots, A^{(n)})$ *are illustrated in Figure 1. In the pointwise application example, we use the function* $\dot{-}$ *defined by* $x \dot{-} y = x - y$ *if* x *and* y *are both real numbers and* $x \geq y$, *and* $x \dot{-} y = 0$ *otherwise.*

The formal semantics of MATLANG expressions is defined in a straightforward manner. Expressions will be evaluated over instances where an *instance* I is a function, defined on a nonempty finite set $\mathrm{var}(I)$ of matrix variables, that assigns a matrix to each element of $\mathrm{var}(I)$. The rules that allow to derive that an expression e, on an instance I, *successfully evaluates* to a matrix A, denoted by $e(I) = A$, is defined recursively as follows. If $M \in \mathrm{var}(I)$, then $M(I) := I(M)$. If $e_1(I) = A$ and $e_2(I[M := A]) = B$, where $I[M := A]$ is the instance obtained from I by mapping M to the matrix A, then $(\mathsf{let}\ M = e_1\ \mathsf{in}\ e_2)(I) := B$. We have $e^*(I) := (e(I))^*$, $(\mathbf{1}(e))(I) := \mathbf{1}(e(I))$, and if $e(I)$ is a column vector, then $(\mathsf{diag}(e))(I) := \mathsf{diag}(e(I))$. Moreover, if the number of columns of $e_1(I)$ is equal to the number of rows of $e_2(I)$, then $(e_1 \cdot e_2)(I) := e_1(I) \cdot e_2(I)$. Finally, if $e_k(I)$ for $k \in \{1, \dots, n\}$ all have the same dimensions, then $\mathsf{apply}[f](e_1, \dots, e_n) := \mathsf{apply}[f](e_1(I), \dots, e_n(I))$.

The reason why an evaluation may not succeed (i.e., $e(I)$ may not be defined) is that diag, apply, and matrix multiplication have conditions on the dimensions of matrices that need to be satisfied for the operations to be well-defined.

EXAMPLE 2.2 (SCALARS). *As a first example we show how to express scalars (elements in* $\mathbf{C}$*). Obviously, in practice, scalars would be part of the language. In this paper, however, we are interested in expressiveness, so we start from a minimal language (*MATLANG*) and then see what is expressible in this language. To express a scalar* $c \in \mathbf{C}$, *consider (by abuse of notation) the constant function* $c : \mathbf{C} \to \mathbf{C} : z \mapsto c$ *and the* MATLANG *expression*

$$c := \mathsf{apply}[c]\big(\mathbf{1}(\mathbf{1}(M)^*)\big).$$

Regardless of the matrix assigned to M, *the expression evaluates to the* 1×1 *matrix whose unique entry is scalar* c.

EXAMPLE 2.3 (SCALAR MULTIPLICATION). *We can also express scalar multiplication of a matrix by a scalar, i.e., the operation which multiplies every entry of a matrix by the same scalar. Indeed, let* c *be a scalar and consider the* MATLANG *expression*

$$\mathsf{let}\ O = \mathbf{1}(M) \cdot c(M) \cdot (\mathbf{1}(M^*))^*\ \mathsf{in}\ \mathsf{apply}[\times](O, M),$$

where c *is the scalar expression from the previous example. If* M *is assigned an* $m \times n$ *matrix* A, *then* $c(A)$ *returns the* 1×1 *matrix* $[c]$ *and in variable* O *we compute the* $m \times n$ *matrix where every entry equals* c. *Then pointwise multiplication* $\times$ *with returns* $x \times y$ *on input* (x, y) *is used to do the scalar multiplication of* A *by* c. *This example generalizes in a straightforward manner to*

$$\mathsf{apply}[\times]\big(\mathbf{1}(e_2) \cdot e_1 \cdot (\mathbf{1}(e_2^*))^*, e_2\big),$$

where e_1 *and* e_2 *are* MATLANG *expressions such that* $e_1(I)$ *is a* 1×1*-matrix for any instance* I. *It should be clear that this expression evaluates to the scalar multiplication of* $e_2(I)$ *by* $e_1(I)$ *for any* I. *We use* $e_1 \odot e_2$ *as a shorthand notation for this expression. For example,* $c \odot e_2$ *represents the scalar multiplication of* e_2 *by the scalar* c.

EXAMPLE 2.4 (GOOGLE MATRIX). *Let* A *be the adjacency matrix of a directed graph (modeling the Web graph) on* n *nodes numbered* $1, \dots, n$. *Let* $0 < d < 1$ *be a fixed "damping factor". Let* k_i *denote the outdegree of node* i. *For simplicity, we assume* k_i *is nonzero for every* i. *Then the Google matrix [9, 6] of* A *is the* $n \times n$ *matrix* G *defined by* $G_{i,j} = dA_{ij}/k_i + (1-d)/n$. *The calculation of* G *from* A *can be expressed in* MATLANG *as follows:*

$$\begin{array}{l} \mathsf{let}\ J = \mathbf{1}(A) \cdot \mathbf{1}(A)^*\ \mathsf{in} \\ \quad \mathsf{let}\ B = \mathsf{apply}[/](A, A \cdot J)\ \mathsf{in} \\ \quad\quad \mathsf{let}\ N = \mathbf{1}(A)^* \cdot \mathbf{1}(A)\ \mathsf{in} \\ \quad\quad\quad \mathsf{apply}[+](d \odot B, (1-d) \odot \big(\mathsf{apply}[1/x](N)\big) \odot J) \end{array}$$

In variable J *we compute the* $n \times n$ *matrix where every entry equals one. In* $A \cdot J$ *we compute the* $n \times n$ *matrix where all entries in the* i*th row equal* k_i. *An* $n \times n$ *matrix holding the entries* A_{ij}/k_i *is computed in* B. *In* N *we compute the* 1×1 *matrix containing the value* n. *The pointwise functions applied are addition, division, and reciprocal. We use the shorthand for constants (*d *and* $1-d$*) from Example 2.2, and* $\odot$ *from Example 2.3.*

2.2 Types and schemas

We now introduce a notion of schema, which assigns types to matrix names, so that expressions can be type-checked against schemas. We already remarked the need for this. Indeed, due to conditions on the dimensions of matrices, MATLANG expressions are not well-defined on all instances. For example, if I is an instance where $I(M)$ is a 3×4 matrix and $I(N)$ is a 2×4 matrix, then the expression $M \cdot N$ is not defined on I. The expression $M \cdot N^*$, however, is well-defined on I.

Our types need to be able to guarantee equalities between numbers of rows or numbers of columns, so that apply and matrix multiplication can be type-checked. Our types also need to be able to recognize vectors, so that diag can be type-checked.

Formally, we assume a sufficient supply of *size symbols*, which we will denote by the letters α, β, γ. A size symbol represents the number of rows or columns of a matrix.

Together with an explicit 1, we can indicate arbitrary matrices as $\alpha \times \beta$, square matrices as $\alpha \times \alpha$, column vectors as $\alpha \times 1$, row vectors as $1 \times \alpha$, and scalars as 1×1. Formally, a *size term* is either a size symbol or an explicit 1. A *type* is then an expression of the form $s_1 \times s_2$ where s_1 and s_2 are size terms. Finally, a *schema* $\mathcal{S}$ is a function, defined on a nonempty finite set $\text{var}(\mathcal{S})$ of matrix variables, that assigns a type to each element of $\text{var}(\mathcal{S})$.

The rules that allow to derive that an expression e over a schema $\mathcal{S}$ *successfully infers* an output type τ, denoted by $\mathcal{S} \vdash e : \tau$, are defined recursively as follows. If $M \in \text{var}(\mathcal{S})$, then $\mathcal{S} \vdash M : \mathcal{S}(M)$. If $\mathcal{S} \vdash e_1 : \tau_1$ and $\mathcal{S}[M := \tau_1] \vdash e_2 : \tau_2$, where $\mathcal{S}[M := \tau]$ denotes the schema that is obtained from $\mathcal{S}$ by mapping M to the type τ, then $\mathcal{S} \vdash \mathsf{let}\ M = e_1\ \mathsf{in}\ e_2 : \tau_2$. If $\mathcal{S} \vdash e : s_1 \times s_2$, then $\mathcal{S} \vdash e^* : s_2 \times s_1$ and $\mathcal{S} \vdash \mathbf{1}(e) : s_1 \times 1$. If $\mathcal{S} \vdash e : s \times 1$, then $\mathcal{S} \vdash \mathsf{diag}(e) : s \times s$. If $\mathcal{S} \vdash e_1 : s_1 \times s_2$ and $\mathcal{S} \vdash e_2 : s_2 \times s_3$, then $\mathcal{S} \vdash e_1 \cdot e_2 : s_1 \times s_3$. Finally, $\mathcal{S} \vdash e_k : \tau$ for $k \in 1, \dots, n$ with $n > 0$ and $f : \mathbf{C}^n \to \mathbf{C}$, then $\mathcal{S} \vdash \mathsf{apply}[f](e_1, \dots, e_n) : \tau$.

When we cannot infer a type, we say e is not well-typed over $\mathcal{S}$. For example, when $\mathcal{S}(M) = \alpha \times \beta$ and $\mathcal{S}(N) = \gamma \times \beta$, then the expression $M \cdot N$ is not well-typed over $\mathcal{S}$. The expression $M \cdot N^*$, however, is well-typed with output type $\alpha \times \gamma$.

To establish the soundness of the type system, we need a notion of conformance of an instance to a schema.

Formally, a *size assignment* σ is a function from size symbols to positive natural numbers. We extend σ to any size term by setting $\sigma(1) = 1$. Now, let $\mathcal{S}$ be a schema and I an instance with $\text{var}(I) = \text{var}(\mathcal{S})$. We say that I is an *instance* of $\mathcal{S}$ if there is a size assignment σ such that for all $M \in \text{var}(\mathcal{S})$, if $\mathcal{S}(M) = s_1 \times s_2$, then $I(M)$ is a $\sigma(s_1) \times \sigma(s_2)$ matrix. In that case we also say that I *conforms* to $\mathcal{S}$ by the size assignment σ.

PROPOSITION 2.5 (SAFETY). *If $\mathcal{S} \vdash e : s_1 \times s_2$, then for every instance I conforming to $\mathcal{S}$, by size assignment σ, the matrix $e(I)$ is well-defined and has dimensions $\sigma(s_1) \times \sigma(s_2)$.*

3. EXPRESSIVE POWER OF MATLANG

3.1 Relational representation of matrices

It is natural to represent an $m \times n$ matrix A by a ternary relation

$$Rel_2(A) := \{(i, j, A_{i,j}) \mid i \in \{1, \dots, m\},\ j \in \{1, \dots, n\}\}.$$

In the special case where A is an $m \times 1$ matrix (column vector), A can also be represented by a binary relation $Rel_1(A) := \{(i, A_{i,1}) \mid i \in \{1, \dots, m\}\}$. Similarly, a $1 \times n$ matrix (row vector) A can be represented by $Rel_1(A) := \{(j, A_{1,j}) \mid j \in \{1, \dots, n\}\}$. Finally, a 1×1 matrix (scalar) A can be represented by the unary singleton relation $Rel_0(A) := \{(A_{1,1})\}$.

Note that in MATLANG, we perform calculations on matrix entries, but not on row or column indices. This fits well to the relational model with aggregates as formalized by Libkin [30]. In this model, the columns of relations are typed as "base", indicated by $\mathbf{b}$, or "numerical", indicated by $\mathbf{n}$. In the relational representations of matrices presented above, the last column is of type $\mathbf{n}$ and the other columns (if any) are of type $\mathbf{b}$. In particular, in our setting, numerical columns hold complex numbers. We now rephrase our relational encoding more formally in this setting.

That is, we assume a supply of *relation variables*, which, for convenience, we can take to be the same as the matrix variables. A *relation type* is a tuple of $\mathbf{b}$'s and $\mathbf{n}$'s. A *relational schema* $\mathcal{S}$ is a function, defined on a nonempty finite set $\text{var}(\mathcal{S})$ of relation variables, that assigns a relation type to each element of $\text{var}(\mathcal{S})$.

To define relational instances, we assume a countably infinite universe **dom** of abstract atomic data elements. For notational convenience, we assume that the natural numbers are contained in **dom**.

Let τ be a relation type. A *tuple of type* τ is a tuple $(t(1), \dots, t(n))$ of the same arity as τ, such that $t(i) \in \mathbf{dom}$ when $\tau(i) = \mathbf{b}$, and $t(i)$ is a complex number when $\tau(i) = \mathbf{n}$. A *relation of type* τ is a finite set of tuples of type τ. An *instance* of a relational schema $\mathcal{S}$ is a function I defined on $\text{var}(\mathcal{S})$ so that $I(R)$ is a relation of type $\mathcal{S}(R)$ for every $R \in \text{var}(\mathcal{S})$.

The matrix data model can now be formally connected to the relational data model, as follows. Let $\tau = s_1 \times s_2$ be a matrix type. Let us call τ a *general type* if s_1 and s_2 are both size symbols; a *vector type* if s_1 is a size symbol and s_2 is 1, or vice versa; and the *scalar type* if τ is 1×1. To every matrix type τ we associate a relation type

$$Rel(\tau) := \begin{cases} (\mathbf{b}, \mathbf{b}, \mathbf{n}) & \text{if } \tau \text{ is a general type;} \\ (\mathbf{b}, \mathbf{n}) & \text{if } \tau \text{ is a vector type;} \\ (\mathbf{n}) & \text{if } \tau \text{ is the scalar type.} \end{cases}$$

Then to every matrix schema $\mathcal{S}$ we associate the relational schema $Rel(\mathcal{S})$ where $Rel(\mathcal{S})(M) = Rel(\mathcal{S}(M))$ for every $M \in \text{var}(\mathcal{S})$. For each instance I of $\mathcal{S}$, we define the instance $Rel(I)$ over $Rel(\mathcal{S})$ by

$$Rel(I)(M) := \begin{cases} Rel_2(I(M)) & \text{if } \mathcal{S}(M) \text{ is a general type;} \\ Rel_1(I(M)) & \text{if } \mathcal{S}(M) \text{ is a vector type;} \\ Rel_0(I(M)) & \text{if } \mathcal{S}(M) \text{ is the scalar type.} \end{cases}$$

3.2 To relational algebra with summation

Given the representation of matrices by relations, we now show that MATLANG can be simulated in the relational algebra with aggregates. Actually, the only aggregate operation we need is summation. The relational algebra with summation extends the well-known relational algebra for relational databases and is defined as follows. For a full formal definition, see [30]. For our purposes it suffices to highlight the following about the relational algebra with summation. Expressions are built up from relation names using the classical operations union, set difference, Cartesian product ($\times$), selection (σ), and projection (π), plus two new operations: *function application* and *summation.* For selection, we only use equality and nonequality comparisons on base columns. No selection on numerical columns will be needed in our setting. Function application and summation are defined as follows.

- For any function $f : \mathbf{C}^n \to \mathbf{C}$, the operation $\mathsf{apply}[f; i_1, \dots, i_n]$ can be applied to any relation r having $\{i_1, \dots, i_n\}$ as a subset of its set of numerical columns. The result is the relation $\{(t, f(t(i_1), \dots, t(i_n))) \mid t \in r\}$, appending a numerical column to r. We allow $n = 0$, i.e., constants f.

- The operation $\mathsf{sum}[i; i_1, \dots, i_n]$ can be applied to any relation r having columns i, i_1, ..., i_n, where column i must be numerical. In our setting we only need the operation in cases where columns $i_1, \ldots, i_n$ are base columns. The result of the operation is the relation

$$\Big\{\big(t(i_1), \ldots, t(i_n), \sum_{t' \in \mathsf{group}[i_1,\ldots,i_n](r,t)} t'(i)\big) \;\Big|\; t \in r\Big\},$$

where $\mathsf{group}[i_1, \ldots, i_n](r, t)$ is equal to

$$\big\{t' \in r \;\big|\; t'(i_1) = t(i_1) \wedge \cdots \wedge t'(i_n) = t(i_n)\big\}.$$

Again, n can be zero, in which case the result is a singleton.

Given that relations are typed, one can define well-typedness for expressions in the relation algebra with summation, and define the output type. We omit this definition here, as it follows a well-known methodology [40] and is analogous to what we have already done for MATLANG in Section 2.2.

THEOREM 3.1. *Let $\mathcal{S}$ be a matrix schema, and let e be a* MATLANG *expression that is well-typed over $\mathcal{S}$ with output type τ. Let $\ell = 2$, 1, or 0, depending on whether τ is general, a vector type, or scalar, respectively.*

1. *There exists an expression $Rel(e)$ in the relational algebra with summation that is well-typed over $Rel(\mathcal{S})$ with output type $Rel(\tau)$ such that for every instance I of $\mathcal{S}$, we have $Rel_\ell(e(I)) = Rel(e)(Rel(I))$.*
2. *The expression $Rel(e)$ uses neither set difference, nor selection conditions on numerical columns.*
3. *The only functions used in $Rel(e)$ are those used in pointwise applications in e; complex conjugation; multiplication of two numbers; and the constant functions* 0 *and* 1.

3.3 To relational calculus with summation

We can sharpen Theorem 3.1 by working in the *relational calculus* with aggregates. In this logic, we have base variables and numerical variables. Base variables can be bound to base columns of relations, and compared for equality. Numerical variables can be bound to numerical columns, and can be equated to function applications and aggregates. We will not recall the syntax formally, see [30] for a full definition. It turns out that when simulating MATLANG expression in the relational calculus with aggregates we only need formulas with at most *three* base variables.

PROPOSITION 3.2. *Let $\mathcal{S}$, e, τ, and ℓ as in Theorem 3.1. For every* MATLANG *expression e there is a formula φ_e over $Rel(\mathcal{S})$ in the relational calculus with summation, such that*

1. *If τ is general, $\varphi_e(i, j, z)$ has two free base variables i and j and one free numerical variable z; if τ is a vector type, we have $\varphi_e(i, z)$; and if τ is scalar, we have $\varphi_e(z)$.*
2. *For every instance I, the relation defined by φ_e on $Rel(I)$ equals $Rel_\ell(e(I))$.*
3. *The formula φ_e uses only three distinct base variables. The functions used in pointwise applications in φ_e are as in the statement of Theorem 3.1. Furthermore, φ_e neither uses equality conditions between numerical variables nor equality conditions on base variables involving constants.*

3.4 Expressing graph queries

We now express relational queries as matrix queries. This works best for binary relations, or graphs, which we can represent by their adjacency matrices.

Formally, we define a *graph schema* to be a relational schema where every relation variable is assigned the type $(\mathbf{b}, \mathbf{b})$ of arity two. We define a *graph instance* as an instance I of a graph schema, where the active domain of I (i.e., the domain elements that occur in some tuple of some relation of I) equals $\{1, \ldots, n\}$ for some positive natural number n.

To every graph schema $\mathcal{S}$ we associate a matrix schema $Mat(\mathcal{S})$, where $(Mat(\mathcal{S}))(R) = \alpha \times \alpha$ for every $R \in \mathrm{var}(\mathcal{S})$, for a fixed size symbol α. So, all matrices are square matrices of the same dimension. Let I be a graph instance of $\mathcal{S}$, with active domain $\{1, \ldots, n\}$. We will denote the $n \times n$ adjacency matrix of a binary relation r over $\{1, \ldots, n\}$ by $Adj_I(r)$. Now any such instance I is represented by the matrix instance $Mat(I)$ over $Mat(\mathcal{S})$, where $Mat(I)(R) = Adj_I(I(R))$ for every $R \in \mathrm{var}(\mathcal{S})$.

A *graph query* over a graph schema $\mathcal{S}$ is a function that maps each graph instance I of $\mathcal{S}$ to a binary relation on the active domain of I. We say that a MATLANG expression e *expresses* the graph query q if e is well-typed over $Mat(\mathcal{S})$ with output type $\alpha \times \alpha$, and for every graph instance I of $\mathcal{S}$, we have $Adj_I(q(I)) = e(Mat(I))$.

We can now give a partial converse to Theorem 3.1. We assume active-domain semantics for first-order logic [1]. Note that the following result deals only with pure first-order logic, without aggregates or numerical columns.

THEOREM 3.3. *Every graph query expressible in* FO^3 *(first-order logic with equality, using at most three distinct variables) is expressible in* MATLANG. *The only functions needed in pointwise applications are boolean functions on $\{0, 1\}$, and testing if a number is positive.*

We can complement the above theorem by showing that the quintessential first-order query requiring *four* variables is not expressible.

PROPOSITION 3.4. *The graph query over a single binary relation R that maps I to $I(R)$ if $I(R)$ contains a four-clique, and to the empty relation otherwise, is not expressible in* MATLANG.

We conclude by showing that MATLANG cannot express the transitive-closure graph query which maps a graph to its transitive closure. This follows from the locality of the calculus with aggregates [30].

PROPOSITION 3.5. *The graph query over a single binary relation R that maps I to the transitive-closure of $I(R)$ is not expressible in* MATLANG.

4. MATRIX INVERSION

We now consider the extension of MATLANG with matrix inversion. Let $\mathcal{S}$ be a schema and e be an expression that is well-typed over $\mathcal{S}$, with output type of the form $\alpha \times \alpha$. Then the expression e^{-1} is also well-typed over $\mathcal{S}$, with the same output type $\alpha \times \alpha$. The semantics is defined as follows. For an instance I, if $e(I)$ is an invertible matrix, then $e^{-1}(I)$ is defined to be the inverse of $e(I)$; otherwise, it is defined to be the zero square matrix of the same dimensions as $e(I)$. The extension of MATLANG with inversion is denoted by MATLANG + inv.

EXAMPLE 4.1 (PAGERANK). *Recall Example 2.4 where we computed the Google matrix of A. In the process we already showed how to compute the $n \times n$ matrix B defined by $B_{i,j} = A_{i,j}/k_i$, and the scalar n. We use e_B and e_n to denote the corresponding* MATLANG *expressions. Let I be the $n \times n$ identity matrix, and let $\mathbf{1}$ denote the $n \times 1$ column vector consisting of all ones. The PageRank vector v of A can be computed as follows [14]:*

$$v = \frac{1-d}{n}(I - dB)^{-1}\mathbf{1}.$$

This calculation is readily expressed in MATLANG + inv *as*

$$(1-d) \odot (\mathsf{apply}[1/x](e_n)) \odot \\ \big(\mathsf{apply}[-](\mathsf{diag}(\mathbf{1}(M)), d \odot e_B)\big)^{-1} \cdot \mathbf{1}(M).$$

EXAMPLE 4.2 (TRANSITIVE CLOSURE). *The reflexive-transitive closure of a binary relation is expressible in* MATLANG + inv. *Let A be the adjacency matrix of a binary relation r on $\{1, \ldots, n\}$. Let I be the $n \times n$ identity matrix, expressible as* $\mathsf{diag}(\mathbf{1}(A))$. *Let e_n be the expression computing the scalar n. The sum of the absolute values of the entries of each column of $B = \frac{1}{n+1}A$ is strictly less than 1, so $S = \sum_{k=0}^{\infty} B^k$ converges, and is equal to $(I - B)^{-1}$ [17, Lemma 2.3.3]. Now (i, j) belongs to the reflexive-transitive closure of r if and only if $S_{i,j}$ is nonzero. Thus, we can compute the reflexive-transitive closure of r by evaluating*

$$\mathsf{let}\ M = \mathsf{apply}[-]\big(\mathsf{diag}(\mathbf{1}(M)), \mathsf{apply}[1/(x{+}1)](e_n) \odot M\big)\ \mathsf{in} \\ \mathsf{apply}[\neq 0](M^{-1})$$

by assigning matrix variable M to A. Here, $\neq 0$ is the function which returns 1 if the value is nonzero and 0 otherwise. We can express the transitive closure by multiplying the above expression by M.

Given our earlier observation that the transitive-closure query cannot be expressed in MATLANG (Proposition 3.5) and the MATLANG + inv expression given in the previous example which does express this query, we may conclude:

THEOREM 4.3. MATLANG+inv *is strictly more powerful than* MATLANG *in expressing graph queries.*

Once we have the transitive closure, we can do many other things such as checking bipartiteness of undirected graphs, checking connectivity, and checking cyclicity. Using Theorem 3.3 one can show that MATLANG is able to reduce these queries to the transitive-closure query.

5. EIGENVECTORS

We next consider the extension of MATLANG with an operation eigen. Formally, we define the operation eigen as follows. Let A be an $n \times n$ matrix. Recall that A is called *diagonalizable* if there exists a basis of $\mathbf{C}^n$ consisting of eigenvectors of A. In that case, there also exists such a basis where eigenvectors corresponding to the same eigenvalue are orthogonal. Accordingly, we define eigen(A) to return an $n \times n$ matrix, the columns of which form a basis of $\mathbf{C}^n$ consisting of eigenvectors of A, where eigenvectors corresponding to a same eigenvalue are orthogonal. If A is not diagonalizable, we define eigen(A) to be the $n \times n$ zero matrix.

Note that eigen is nondeterministic; in principle there are infinitely many possible results. This models the situation in practice where numerical packages such as R or MATLAB return approximations to the eigenvalues and a set of corresponding eigenvectors. Eigenvectors, however, are not unique. In fact, there are infinitely many eigenvectors.

Hence, some care must be taken in extending MATLANG with the eigen operation. Syntactically, as for inversion, whenever e is a well-typed expression with a square output type, we now also allow the expression eigen(e), with the same output type. Semantically, however, the semantic rules of MATLANG must be adapted so that they do not infer statements of the form $e(I) = B$, but rather of the form $B \in e(I)$, i.e., B is a possible result of $e(I)$. The let-construct now becomes crucial; it allows us to assign a possible result of eigen to a new variable, and work with that intermediate result consistently.

EXAMPLE 5.1 (RANK OF A MATRIX). *First, we remark that one can show that the diagonal matrix containing the eigenvalues Λ corresponding to the matrix B of eigenvectors computed by* eigen(A) *is expressible in* MATLANG + eigen. *Hence we allow a shorthand notation where* eigen(A) *obtains the tuple (B, Λ) instead of just B. We then agree that Λ, like B, is a zero matrix if A is not diagonalizable.*

Since the rank of a diagonalizable matrix equals the number of nonzero entries in its diagonal form, we can express the rank of a diagonalizable matrix A as follows:

$$\mathsf{let}\ (B, \Lambda) = \mathsf{eigen}(A)\ \mathsf{in}\ \mathbf{1}(A)^* \cdot \mathsf{apply}[\neq 0](\Lambda) \cdot \mathbf{1}(A).$$

Using a known argument from linear algebra we obtain that MATLANG + inv is subsumed by MATLANG + eigen.

THEOREM 5.2. *Matrix inversion is expressible in* MATLANG + eigen.

An interesting open problem is the following: *Are there graph queries expressible deterministically in* MATLANG + eigen*, but not in* MATLANG + inv*?*

6. THE EVALUATION PROBLEM

We next consider the evaluation problem of expressions in our most expressive language MATLANG+eigen. Naively, the evaluation problem asks, given an input instance I and an expression e, to compute the result $e(I)$. There are some issues with this naive formulation, however. Indeed, in our theory we have been working with arbitrary complex numbers. How do we even represent the input? Notably, the eigen operation on a matrix with only rational entries may produce irrational entries. In fact, the eigenvalues of an adjacency matrix (even of a tree) need not even be definable in radicals [16]. Practical systems, of course, apply techniques from numerical mathematics to compute rational approximations. But it is still theoretically interesting to consider the exact evaluation problem. For a treatise on computations of eigenvectors, inverses, and other matrix notions, we refer to [17].

Our approach is to represent the output symbolically, following the idea of constraint query languages [23, 28]. Specifically, we can define the input-output relation of an expression, for given dimensions of the input matrices, by an *existential first-order logic formula over the reals.* Such formulas are built from real variables, integer constants, addition, multiplication, equality, inequality ($<$), disjunction, conjunction, and existential quantification.

Any $m \times n$ matrix A can be represented by a tuple of $2mn$ real numbers. Indeed, let $a_{i,j} = \Re A_{i,j}$ (the real part of a complex number), and let $b_{i,j} = \Im A_{i,j}$ (the imaginary part). Then A can be represented by the tuple $(a_{1,1}, b_{1,1}, a_{1,2}, b_{1,2}, \ldots, a_{m,n}, b_{m,n})$. The next result introduces the variables $x_{M,i,j,\Re}$, $x_{M,i,j,\Im}$, $y_{i,j,\Re}$, and $y_{i,j,\Im}$, where the x-variables describe an arbitrary input matrix $I(M)$ and the y-variables describe an arbitrary possible output matrix $e(I)$.

In the following, an *input-sized expression* consists of a schema $\mathcal{S}$, an expression e in $\mathsf{MATLANG} + \mathsf{eigen}$ that is well-typed over $\mathcal{S}$ with output type $t_1 \times t_2$, and a size assignment σ defined on the size symbols occurring in $\mathcal{S}$. For complexity considerations, we assume the sizes given in σ are coded in unary.

THEOREM 6.1. *There exists a polynomial-time computable translation that maps any input-sized expression e to an existential first-order formula ψ_e over the vocabulary of the reals, expanded with symbols for the functions used in pointwise applications in e, such that*

1. *Formula ψ_e has the following free variables:*
   - *For every $M \in \mathrm{var}(\mathcal{S})$, let $\mathcal{S}(M) = s_1 \times s_2$. Then ψ_e has the free variables $x_{M,i,j,\Re}$ and $x_{M,i,j,\Im}$, for $i = 1, \ldots, \sigma(s_1)$ and $j = 1, \ldots, \sigma(s_2)$.*
   - *In addition, ψ_e has the free variables $y_{e,i,j,\Re}$ and $y_{e,i,j,\Im}$, for $i = 1, \ldots, \sigma(t_1)$ and $j = 1, \ldots, \sigma(t_2)$.*

   The set of these free variables is denoted by $\mathrm{FV}(\mathcal{S}, e, \sigma)$.
2. *Any assignment ρ of real numbers to these variables specifies, through the x-variables, an instance I conforming to $\mathcal{S}$ by σ, and through the y-variables, a $\sigma(t_1) \times \sigma(t_2)$ matrix B.*
3. *Formula ψ_e is true over the reals under such an assignment ρ, if and only if $B \in e(I)$.*

The existential theory of the reals is decidable; actually, the full first-order theory of the reals is decidable [3, 4]. But, specifically the class of problems that can be reduced in polynomial time to the existential theory of the reals forms a complexity class on its own, known as $\exists\mathbf{R}$ [36, 37]. This class lies between NP and PSPACE. The above theorem implies that the *intensional evaluation problem for* $\mathsf{MATLANG} + \mathsf{eigen}$ belongs to this complexity class. We define this problem as follows. The idea is that an arbitrary specification, expressed as an existential formula χ over the reals, can be imposed on the input-output relation of an input-sized expression.

DEFINITION 6.2. *The* intensional evaluation problem *is a decision problem that takes as input: (1) an input-sized expression $(\mathcal{S}, e, \sigma)$, where all functions used in pointwise applications are explicitly defined using existential formulas over the reals, and (2) an existential formula χ with free variables in* $\mathrm{FV}(\mathcal{S}, e, \sigma)$.

The problem asks if there exists an instance I conforming to $\mathcal{S}$ by σ and a matrix $B \in e(I)$ such that (I, B) satisfies χ.

An input $(\mathcal{S}, e, \sigma, \chi)$ is a yes-instance to the intensional evaluation problem precisely when the existential sentence $\exists \mathrm{FV}(\mathcal{S}, e, \sigma)(\psi_e \wedge \chi)$ is true in the reals, where ψ_e is the formula obtained by Theorem 6.1. Hence we can conclude:

COROLLARY 6.3. *The intensional evaluation problem for* $\mathsf{MATLANG} + \mathsf{eigen}$ *belongs to $\exists\mathbf{R}$.*

Since the full first-order theory of the reals is decidable, our theorem implies many other decidability results, including that both the equivalence problem and the determinacy problem for input-sized expressions are decidable.

Corollary 6.3 gives an $\exists\mathbf{R}$ upper bound on the combined complexity of query evaluation [41]. Our final result is a matching lower bound, already for data complexity alone.

THEOREM 6.4. *There exists a fixed schema $\mathcal{S}$ and a fixed expression e in* $\mathsf{MATLANG} + \mathsf{eigen}$*, well-typed over $\mathcal{S}$, such that the following problem is hard for $\exists\mathbf{R}$: Given an integer instance I over $\mathcal{S}$, decide whether the zero matrix is a possible result of $e(I)$. The pointwise applications in e use only simple functions definable by quantifier-free formulas over the reals.*

7. CONCLUSION

There is a commendable trend in contemporary database research to leverage and considerably extend techniques from database query processing and optimization to support large-scale linear algebra computations. In principle, data scientists could then work directly in SQL or related languages. Still, some users will prefer to continue using the matrix languages they are more familiar with. Supporting these languages is also important so that existing code need not be rewritten.

From the perspective of database theory, it then becomes relevant to understand the expressive power of these languages as well as possible. In this paper we have proposed a framework for viewing matrix manipulation from the point of view of expressive power of database query languages. Our results formally confirm that the basic set of matrix operations offered by systems in practice, formalized here in the language $\mathsf{MATLANG} + \mathsf{inv} + \mathsf{eigen}$, really is adequate for expressing a range of linear algebra techniques and procedures.

Deep inexpressibility results have been developed for logics with rank operators [32]. Although these results are mainly concerned with finite fields, they might still provide valuable insight in our open questions. Also, we have not covered all standard constructs from linear algebra. For instance, it may be worthwhile to extend our framework with the operation of putting matrices in upper triangular form, with the Gram-Schmidt procedure (which is now partly hidden in the eigen operation), and with the singular value decomposition.

There also have been proposals to go beyond matrices, introducing data models and algebra for tensors or multidimensional arrays [33, 24, 34]. It would be interesting to understand the expressive power of such tensor languages.

Acknowledgments.
We thank Bart Kuijpers, Lauri Hella, Wied Pakusa, Christoph Berkholz, and Anuj Dawar for helpful discussions, and Wim Martens for useful comments on the text. R.B. is a postdoctoral fellow of the Research Foundation – Flanders (FWO).

8. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] M. Abo Khamis, H. Ngo, and A. Rudra. FAQ: questions asked frequently. In *Proc. PODS 2016*. ACM Press, 2016.

[3] D. Arnon. Geometric reasoning with logic and algebra. *Artif. Intell.*, 37:37–60, 1988.

[4] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*. Springer, second edition, 2008.

[5] M. Boehm et al. SystemML: declarative machine learning on Spark. *Proc. VLDB Endow*, 9(13):1425–1436, 2016.

[6] A. Bonato. *A Course on the Web Graph*, volume 89 of *Graduate Studies in Mathematics*. American Mathematical Society, 2008.

[7] R. Brijder, F. Geerts, J. Van den Bussche, and T. Weerwag. On the expressive power of query languages for matrices. *ACM TODS*, 2019. To appear.

[8] R. Brijder, M. Gyssens, and J. Van den Bussche. On matrices and K-relations. arXiv:1904.03934, 2019.

[9] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Comput. Networks ISDN*, 30:107–117, 1998.

[10] L. Chen, A. Kumar, J. Naughton, and J. Patel. Towards linear algebra over normalized data. *Proc. VLDB Endow*, 10(11):1214–1225, 2017.

[11] S. Datta, R. Kulkarni, A. Mukherjee, T. Schwentick, and T. Zeume. Reachability is in DynFO. *J. ACM*, 65(5):33:1–33:24, 2018.

[12] A. Dawar. On the descriptive complexity of linear algebra. In W. Hodges and R. de Queiroz, editors, *Proc. WoLLIC 2008*, volume 5110 of *LNCS*, pages 17–25. Springer, 2008.

[13] A. Dawar, M. Grohe, B. Holm, and B. Laubner. Logics with rank operators. In *Proc. LICS 2009*, pages 113–122, 2009.

[14] G. Del Corso, A. Gulli, and F. Romani. Fast PageRank computation via a sparse linear system. *Internet Math.*, 2(3):251–273, 2005.

[15] F. Geerts. On the expressive power of linear algebra on graphs. In *Proc. ICDT 2019*, volume 127 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019.

[16] C. Godsil. Some graphs with characteristic polynomials which are not solvable by radicals. *J. Graph Theory*, 6:211–214, 1982.

[17] G. Golub and C. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, fourth edition, 2013.

[18] T. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *Proc. PODS 2007*, pages 31–40. ACM Press, 2007.

[19] J. Hellerstein et al. The MADlib analytics library: Or MAD skills, the SQL. *Proc. VLDB Endow*, 5(12):1700–1711, 2012.

[20] B. Holm. *Descriptive Complexity of Linear Algebra*. PhD thesis, University of Cambridge, 2010.

[21] D. Hutchison, B. Howe, and D. Suciu. LaraDB: a minimalist kernel for linear and relational algebra computation. In *Proc. BeyondMR 2007*, pages 2:1–2:10, 2007.

[22] K. Iverson. *A Programming Language*. John Wiley & Sons, Inc., 1962.

[23] P. Kanellakis, G. Kuper, and P. Revesz. Constraint query languages. *J. Comput. Syst. Sci.*, 51(1):26–52, Aug. 1995.

[24] M. Kim. *TensorDB and Tensor-Relational Model for Efficient Tensor-Relational Operations*. PhD thesis, Arizona State University, 2014.

[25] A. Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *J. ACM*, 29(3):699–717, 1982.

[26] P. Kolaitis. On the expressive power of logics on finite models. In *Finite Model Theory and Its Applications*, chapter 2. Springer, 2007.

[27] A. Kunft, A. Alexandrov, A. Katsifodimos, and V. Markl. Bridging the gap: Towards optimization across linear and relational algebra. In *Proc. BeyondMR 2016*, pages 1:1–1:4, 2016.

[28] G. Kuper, L. Libkin, and J. Paredaens, editors. *Constraint Databases*. Springer, 2000.

[29] B. Laubner. *The Structure of Graphs and New Logics for the Characterization of Polynomial Time*. PhD thesis, Humboldt-Universität zu Berlin, 2010.

[30] L. Libkin. Expressive power of SQL. *Theor. Comput. Sci.*, 296:379–404, 2003.

[31] S. Luo, Z. Gao, M. Gubanov, L. L. Perez, and C. Jermaine. Scalable linear algebra on a relational database system. In *Proc. ICDE 2017*, pages 523–534. IEEE Computer Society, 2017.

[32] W. Pakusa. *Linear Equation Systems and the Search for a Logical Characterisation of Polynomial Time*. PhD thesis, RWTH Aachen, 2015.

[33] F. Rusu and Y. Cheng. A survey on array storage, query languages, and systems. arXiv:1302.0103, 2013.

[34] T. Sato. Embedding Tarskian semantics in vector spaces. arXiv:1703.03193, 2017.

[35] T. Sato. A linear algebra approach to datalog evaluation. *Theory Pract. Log. Prog.*, 17(3):244–265, 2017.

[36] M. Schaefer. Complexity of some geometric and topological problems. In D. Eppstein and E. Gansner, editors, *Graph Drawing*, volume 5849 of *LNCS*, pages 334–344. Springer, 2009.

[37] M. Schaefer and D. Štefankovič. Fixed points, Nash equilibria, and the existential theory of the reals. *Theory Comput. Syst.*, 60(2):172–193, 2017.

[38] M. Schleich, D. Olteanu, and R. Ciucanu. Learning linear regression models over factorized joins. In *Proc. SIGMOD 2016*, pages 3–18. ACM, 2016.

[39] A. Thomas and A. Kumar. A comparative evaluation of systems for scalable linear algebra-based analytics. *Proc. VLDB Endow*, 11(13):2168–2182, 2018.

[40] J. Van den Bussche, D. Van Gucht, and S. Vansummeren. A crash course in database queries. In *Proc. PODS 2007*, pages 143–154. ACM Press, 2007.

[41] M. Vardi. The complexity of relational query languages. In *Proc. STOC 1982*, pages 137–146, 1982.

[42] Y. Zhang, W. Zhang, and J. Yang. I/O-efficient statistical computing with RIOT. In *Proc. ICDE 2010*, pages 1157–1160, 2010.

Technical Perspective: Online Model Management via Temporally Biased Sampling

Ke Yi
HKUST
yike@ust.hk

Randoms sampling from data streams is a problem with a long history of studies, starting from the famous *reservoir sampling* algorithm that is at least 50 years old [2]. The reservoir sampling algorithm maintains a random sample over all data items that have ever been received from the stream. This is not suitable for many of today's applications on evolving data streams, where recent data is more important than older ones.

There are two popular approaches to dealing with evolving data streams in the literature.

Sliding windows: In the sliding window model, only data that have arrived in the window $[now - w, now]$ are relevant, where *now* is the current time instance, and w is the length (in terms of time) of the window. In the context of random sampling, this means that the sample should be only drawn from data items in the window, each with equal probability. Random sampling in the sliding window model have been well studied, and optimal algorithms are available [1].

Time decay: In the time decay model, the "importance" assigned to each data item decreases as its age. The importance function, in general, can be an arbitrary non-increasing function of age, but the mostly commonly used one is *exponential decay*, where the importance is of an item x is $e^{-\lambda(now-t_x)}$, where t_x is the timestamp of x, and λ is a parameter that controls the rate of the decay. In the context of random sampling, this means that the probability of each item being sampled should be proportional to its importance.

The following figure illustrates the difference between the sliding window model and the time decay model. While the sliding window model applies a sharp threshold (the length of the sliding window) on the age of the items, the time decay model is much "smoother". It gives everyone a chance to be sampled, although older items have exponentially smaller probabilities to get into the sample.

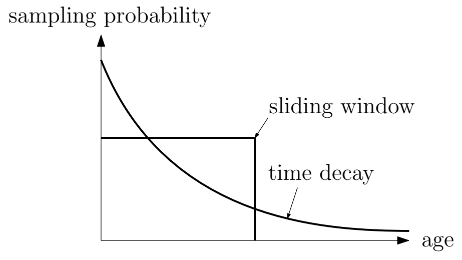


Figure 1: Sampling probability vs. age in the two models.

The following paper by Hentschel, Haas, and Tian takes the time decay approach to random sampling. Building upon prior work, they designed two elegant algorithms with strong theoretical guarantees. The first one, called T-TBS, is very simple to implement and highly scalable, but assumes the arriving batch sizes are i.i.d. with a common mean. The sample size may not be guaranteed when this assumption fails. The second algorithm, called R-TBS, is more complicated, but offers stronger guarantees. It provides a guaranteed upper bound on the sample size, and allows unknown, varying arrival rates. The authors have also applied their time-decayed samples to training machine learning models over evolving data. The results demonstrate promising results showing that these samples can help to refresh the models to capture evolving patterns in the stream more accurately. The paper should be useful for anyone who is interested in random sampling or data analytics over evolving data in general.

1. REFERENCES

[1] V. Braverman, R. Ostrovsky, and C. Zaniolo. Optimal sampling for sliding windows. In *Proc. ACM Symposium on Principles of Database Systems*, 2009.

[2] D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 1st edition, 1969.

Online Model Management via Temporally Biased Sampling

Brian Hentschel
Harvard University
bhentschel@g.harvard.edu

Peter J. Haas
University of Massachusetts
phaas@cs.umass.edu

Yuanyuan Tian
IBM Research – Almaden
ytian@us.ibm.com

ABSTRACT

To maintain the accuracy of supervised learning models in the presence of evolving data streams, we provide temporally-biased sampling schemes that weight recent data most heavily, with inclusion probabilities for a given data item decaying exponentially over time. We then periodically retrain the models on the current sample. We provide and analyze both a simple sampling scheme (T-TBS) that probabilistically maintains a target sample size and a novel reservoir-based scheme (R-TBS) that is the first to provide both control over the decay rate and a guaranteed upper bound on the sample size. The R-TBS and T-TBS schemes are of independent interest, extending the known set of unequal-probability sampling schemes. We discuss distributed implementation strategies; experiments in Spark show that our approach can increase machine learning accuracy and robustness in the face of evolving data.

1. INTRODUCTION

A key challenge for machine learning (ML) is to keep ML models from becoming stale in the presence of evolving data. In the context of the emerging Internet of Things (IoT), for example, the data comprise dynamically changing sensor streams, and a failure to adapt to changing data can lead to a loss of predictive power.

One way to deal with this problem is to re-engineer existing static supervised learning algorithms to become adaptive. Some parametric methods such as regression models can indeed be re-engineered so that the parameters are time-varying, but for many popular non-parametric algorithms such as k-nearest neighbors (kNN) classifiers, decision trees, random forests, gradient boosted machines, and so on, it is not at all clear how re-engineering can be accomplished. The 2017 Kaggle Data Science Survey [1] indicates that a substantial portion of the models that developers use in industry are non-parametric. We therefore consider alternative approaches in which we periodically retrain ML models, allowing static ML algorithms to be used in dynamic settings essentially as-is. There are several possible retraining approaches.

Retraining on cumulative data: Periodically retraining a model on all of the data that has arrived so far is clearly infeasible because of the huge volume of data involved. Moreover, recent data is swamped by the massive amount of past data, so the retrained model is not sufficiently adaptive.

Sliding windows: A simple sliding-window approach periodically retrains on the data from, e.g., the last two hours. If the data arrival rate is high and there is no bound on memory, then one must deal with long retraining times caused by large amounts of data in the window. The simplest way to bound the window size is to retain the last n items. Alternatively, one could try to subsample within the time-based window [10]. The fundamental problem with all of these bounding approaches is that old data is completely forgotten; the problem is especially severe when the data arrival rate is high. This can undermine the robustness of an ML model in situations where old patterns can reassert themselves. For example, a singular event such as a holiday, stock market drop, or terrorist attack can temporarily disrupt normal data patterns, which will reestablish themselves once the effect of the event dies down. Periodic data patterns can lead to the same phenomenon. Another example, from [15], concerns influencers on Twitter: a prolific tweeter might temporarily stop tweeting due to travel, illness, or some other reason, and hence be completely forgotten in a sliding-window approach. Indeed, in real-world Twitter data, almost a quarter of top influencers were of this type, and were missed by a sliding window approach.

Temporally biased sampling: An appealing alternative is a temporally biased sampling-based approach, i.e., maintaining a sample that heavily emphasizes recent data but also contains a small amount of older data, and periodically retraining a model on the sample. By using a time-biased sample, the retraining costs can be held to an acceptable level while not sacrificing robustness in the presence of recurrent patterns. This approach was proposed in [15] in the setting of graph analysis algorithms, and has recently been adopted in the MacroBase system [5]. The orthogonal problem of choosing when to retrain a model is also an important question, and is related to, e.g., the literature on "concept drift" [9]; in this paper we focus on the problem of how to efficiently maintain a time-biased sample.

In more detail, our time-biased sampling algorithms ensure that the "appearance probability" for a given data item, i.e., the probability that the item appears in the current sample, decays over time at a controlled exponential rate. We assume that items arrive in *batches* $\mathcal{B}_1, \mathcal{B}_2, \ldots$, at time points $t_1, t_2, \cdots$, where each batch contains 0 or more items. Our goal is to generate a sequence $\{S_k\}_{k\geq 1}$, where S_k is a sample of the items that have arrived at or prior to time t_k.

These samples should be biased towards recent items, in the following sense. For $1 \leq i \leq k$, denote by $\alpha_{i,k} = t_k - t_i$ the *age* at time t_k of an item belonging to batch $\mathcal{B}_i$. Then for arbitrary times $t_i \leq t_j \leq t_k$ and items $x \in \mathcal{B}_i$ and $y \in \mathcal{B}_j$,

$$\Pr[x \in S_k] / \Pr[y \in S_k] = f(\alpha_{i,k})/f(\alpha_{j,k}) = e^{-\lambda(t_j - t_i)}, \quad (1)$$

where $f(\alpha) = e^{-\lambda\alpha}$ is the exponential *decay function.* (We briefly discuss other decay functions in Section 7.) Thus items with a given timestamp are sampled uniformly, and items with different timestamps are handled in a carefully controlled manner, such that the appearance probability for an item of age α is proportional to $f(\alpha)$. The criterion in (1), which is expressed in terms of wall-clock time, is natural and appealing in applications and, importantly, is interpretable and understandable to users. As discussed in [7, 15], the decay function can be chosen to meet application-specific criteria. If training data is available, λ can also be chosen to maximize accuracy of a specified ML model via cross validation combined with grid search—in our experiments, we found empirically that accuracy tended to be a quasiconvex function of λ, which bodes well for automatic optimization methods such as stochastic gradient descent.

Prior work: It is surprisingly hard to both enforce (1) and to bound the sample size. As discussed in detail in an extended version of this paper [12], prior algorithms that bound the sample size either cannot consistently enforce (1) or cannot handle wall-clock time. Examples of the former include algorithms based on the A-Res scheme of Efraimidis and Spirakis [8] and on Chao's algorithm [6]. A-Res enforces conditions on the *acceptance* probabilities of items; this leads to appearance probabilities that, unlike (1), are both hard to compute and not intuitive. In [12], we show that Chao's algorithm fails to enforce (1) either when initially filling up an empty sample or in the presence of data that arrives slowly relative to the decay rate. The second type of algorithm, due to Aggarwal [4], can only control appearance probabilities based on the indices of the data items and not wall-clock time; this can be suboptimal in the presence of time-varying arrival rates. Thus our new sampling schemes are interesting in their own right, significantly expanding the set of unequal-probability sampling techniques.

T-TBS: We first provide and analyze Targeted-Size Time-Biased Sampling (T-TBS), a relatively simple algorithm that generalizes the Bernoulli sampling scheme in [15] (which we call B-TBS). T-TBS allows complete control over the decay rate (expressed in wall-clock time) and probabilistically maintains a target sample size. T-TBS is easy to implement and highly scalable when applicable, but only works under the strong restriction that the mean sizes of the arriving batches are constant over time and known a priori. There are scenarios where T-TBS might be a good choice (see Section 3), but many applications have non-constant, unknown mean batch sizes and/or cannot tolerate sample overflows.

R-TBS: We then provide a novel algorithm, Reservoir-Based Time-Biased Sampling (R-TBS), that is the first to simultaneously enforce (1) at all times, provide a guaranteed upper bound on the sample size, and allow unknown, varying data arrival rates. Guaranteed bounds are desirable because they avoid memory management issues associated with sample overflows, especially when large numbers of samples are being maintained—so that the probability of *some* sample overflowing is high—or when sampling is being performed in a limited memory setting such as at the "edge" of the IoT. Also, bounded samples reduce variability in retraining times and do not impose upper limits on the incoming data flow.

Distributed implementation: Both T-TBS and R-TBS can be parallelized. Whereas T-TBS is relatively straightforward to implement, an efficient distributed implementation of R-TBS is nontrivial. We exploit various implementation strategies to minimize I/O, avoid unnecessary concurrency control, and make decentralized decisions about which items to insert into, or delete from, the reservoir. Our experiments (Section 6) demonstrate the efficiency and effectiveness of our techniques.

2. BACKGROUND

For the remainder of the paper, assume that batches arrive at regular time intervals, so that $t_i = i\Delta$ for some $\Delta > 0$. All items that arrive in an interval $\big((k-1)\Delta, k\Delta\big]$ are treated as if they arrived at time $k\Delta$, i.e., at the end of the interval, so that all items in batch $\mathcal{B}_i$ have time stamp $i\Delta$. It follows that the age at time t_k of an item that arrived at time $t_i \leq t_k$ is simply $\alpha_{i,k} = (k-i)\Delta$.

In this section, we briefly review two classical sampling schemes whose properties we will combine in the R-TBS algorithm. A detailed description of the two algorithms can be found in [12].

Bernoulli Time-Biased Sampling (B-TBS): One simple sampling scheme [15] processes each incoming batch by first downsampling the current sample and then accepting all items in the batch into the sample with probability 1. Downsampling is accomplished by flipping a coin independently for each item in the sample: an item is retained in the sample with probability $p = e^{-\lambda\Delta}$ and removed with probability $1-p$. In [12], we prove that B-TBS enforces the relation in (1) as required. Unfortunately, the user cannot independently control the expected sample size, which is completely determined by λ and the sizes of the incoming batches.

Batched Reservoir Sampling (B-RS): The standard reservoir sampling algorithm with sample size n accepts the first n items into the sample with probability 1. For $k > n$, the kth item is accepted with probability n/k, overwriting a random victim; in [12], we show how to modify the algorithm to handle batch arrivals. Although B-RS guarantees an upper bound on the sample size, it does not support time biasing. R-TBS (Section 4) maintains a bounded reservoir as in B-RS while supporting time-biased sampling as in B-TBS.

3. TARGETED-SIZE TBS

We now describe the T-TBS scheme, which improves upon the simple Bernoulli sampling scheme B-TBS by ensuring the inclusion property in (1) while providing probabilistic guarantees on the sample size. We write $B_k = |\mathcal{B}_k|$ for $k \geq 1$, and assume that the batch sizes $\{B_k\}_{k\geq 1}$ are independent and identically distributed (i.i.d.) as a random variable B, with common mean $b = \mathrm{E}[B] < \infty$.

The Algorithm: The idea behind the algorithm is to downsample to remove older items, as in B-TBS, but to also downsample the incoming batches at a rate q such that n becomes the "equilibrium" sample size, while also ensuring that (1) holds. Setting $p = e^{-\lambda\Delta}$ as before, a simple calculation shows that, for any $q \in (0,1]$, $\Pr[x \in S_k] = qp^{k-i} = qe^{-\lambda(t_k - t_i)}$, and (1) follows immediately.

To choose q, suppose that $|S_{k-1}| = n$ and we are about to process batch $\mathcal{B}_k$. The expected number of items that will be removed is $(1-p)n$ and the expected number of accepted items is qb. For n to be an equilibrium point, we equate these terms and solve for q to obtain $q = n(1-p)/b$. Note that, even if we always accept all items in an arriving batch (i.e., $q = 1$) but the resulting expected inflow b is less than the expected outflow $n(1-p)$, the sample will consistently fall below n, and so we require that $b \geq n(1-p)$.

ALGORITHM 1: Targeted-size TBS (T-TBS)

```
1  $p = e^{-\lambda\Delta}$: decay factor
2  $n$: target sample size
3  $b$: assumed mean batch size such that $b \geq n(1-p)$
4  Initialize: $S \leftarrow \emptyset$; $q \leftarrow n(1-p)/b$
5  for $k \leftarrow 1, 2, \ldots$ do
6      $m \leftarrow$ Binomial$(|S|, p)$                //simulate $|S|$ trials
7      $S \leftarrow$ Sample$(S, m)$                  //retain $m$ random elements
8      $l \leftarrow$ Binomial$(|\mathcal{B}_k|, q)$              //simulate $|\mathcal{B}_k|$ trials
9      $\mathcal{B}'_k \leftarrow$ Sample$(\mathcal{B}_k, l)$              //downsample new batch
10     $S \leftarrow S \cup \{\mathcal{B}'_k\}$                 //add new items to sample
11     output $S$
```

The resulting sampling scheme is given as Algorithm 1; it precisely controls inclusion probabilities in accordance with (1) while constantly pushing the sample size toward the target value n. Conceptually, at each time t_k, T-TBS first downsamples the current sample by independently flipping a coin for each item with retention probability p. T-TBS then downsamples the arriving batch $\mathcal{B}_k$ via independent coin flips; an item in $\mathcal{B}_k$ is inserted into the sample with probability q. For efficiency, the algorithm exploits the fact that for j independent trials, each having success probability r, the total number of successes has a binomial distribution with parameters j and r. Thus, in lines 6 and 8, the algorithm simulates the coin tosses by directly generating the number of successes m or l—which can be done using standard algorithms [13]—and then retaining m or l randomly chosen items. So the function Binomial(j, r) returns a random sample from the binomial distribution with j independent trials and success probability r per trial, and the function Sample(A, m) returns a uniform random sample, without replacement, containing $\min(m, |A|)$ elements of the set A; note that Sample(A, m) returns an empty sample if $A = \emptyset$, $m = 0$, or both.

Sample-Size Properties: Theorem 1 below precisely describes the sample size behavior of T-TBS, which directly impacts memory requirements, efficiency of memory usage, and ML model retraining time; see [12] for a statement and proof of this result in the setting of general decay functions. (The proof exploits the fact that $\{|S_k|\}_{k\geq 0}$ is a Markov chain.) Denote by $C_k = |S_k|$ the sample size at time t_k and by $\bar{b} \geq 1$ the maximum possible batch size, so that $\Pr[B \leq \bar{b}] = 1$. Also set $\sigma^2 = bq(1+p-q)/(1-p^2) \in (0, \infty)$ and write $p = e^{-\lambda\Delta}$ as before. Write "i.o." to denote that an event occurs "infinitely often", i.e., for infinitely many values of k, and write "w.p.1" for "with probability 1".

Assertions (i)–(iii) of Theorem 1 deal with the distribution of the sample size after a large number of batches have been processed. Specifically, by (i) and (ii), the expected sample size $\mathrm{E}[C_k]$ approximately equals the target size n and the variance of C_k approximately equals the finite constant σ^2, which depends on b, p, and q; note that the convergence of $\mathrm{E}[C_k]$ to n happens exponentially fast. By (iii), the probability that the sample size deviates from n by more than $100\epsilon\%$ is exponentially small when k or n is large, provided that the batch sizes are bounded.

Whereas Assertions (i)–(iii) describe average behavior over many sampling runs, Assertions (iv) and (v) concern the behavior of the successive sample sizes during an individual sampling run. By (iv), any sample size can be attained with positive probability, so one potential type of bad behavior might occur if, with positive probability, the sample size is unstable in that it drifts off to $+\infty$ over time. By (v), if the batch sizes are bounded, then such unstable behavior is ruled out: with probability 1, the sample-size process is stable in that every possible sample-size value occurs infinitely often, with finite expected time between visits. Moreover, the average sample size—averaged over times $t_1, t_2, \ldots, t_k$—converges to n with probability 1 as k becomes large. On the negative side, it follows that, for a given sampling run, the sample size will repeatedly—though infrequently, since the expected sample size at any time point is finite—become arbitrarily large, even if the average behavior is good. This result shows that, even in the most stable case, the sample-size control provided by T-TBS is incomplete, and thus motivates the more complex R-TBS algorithm given in the next section. This sample-size fragility is amplified when batch sizes fluctuate in a non-predicable way, as often happens in practice, and T-TBS can break down; see Section 6.

Theorem 1. *If $\{B_k\}_{k\geq 1}$ are i.i.d. with mean $b < \infty$, then*

(i) $\mathrm{E}[C_k] = n(1-p^k) \uparrow n$ *as* $k \to \infty$*;*

(ii) $\mathrm{Var}[C_k] \to \sigma^2$ *as* $k \to \infty$*;*

(iii) if $\bar{b} < \infty$*, then (a)* $\Pr[C_k \geq (1+\epsilon)n] \leq e^{-O(kn^2\epsilon^2)}$ *for all* $\epsilon, k > 0$ *and (b)* $\Pr[C_k \leq (1-\epsilon)n] \leq e^{-O(kn^2)}$ *for any* $\epsilon \in (0,1)$ *and sufficiently large* k*;*

(iv) $\forall m \geq 0$, $\exists k \geq 0$ *such that* $\Pr[C_k \geq m] > 0$*;*

(v) if $\bar{b} < \infty$*, then (a)* $\Pr[C_k = m \text{ i.o.}] = 1$ *for all* $m \geq 0$*, (b) the expected times between successive visits to state* m *are uniformly bounded for any* $m \geq 0$*, and (c)* $\lim_{k\to\infty}(1/k)\sum_{i=0}^{k} C_i = n$ *w.p.1.*

Despite the fluctuations in sample size, T-TBS is of interest because, when the mean batch size is known and constant over time, and when some sample overflows are tolerable, T-TBS is relatively simple to implement and parallelize, and is very fast (see Section 6). For example, if the data comes from periodic polling of a set of robust sensors, the data arrival rate will be known a priori and will be relatively constant, except for the occasional sensor failure, and hence T-TBS might be appropriate.

4. RESERVOIR-BASED TBS

Our new reservoir-based time-biased sampling algorithm (R-TBS) combines the best features of T-TBS and B-RS, controlling the decay rate while ensuring that the sample never overflows. Importantly, unlike T-TBS, the R-TBS algorithm can handle any sequence of batch sizes. The proofs of all the theorems in this section can be found in [12].

4.1 Item Weights and Latent Samples

To precisely control the sample size in the presence of decay, we essentially need to handle samples having "fractional size". We do this via "item weights" and "latent samples".

Item weights: In R-TBS, the *weight* of an item of age α is given by $f(\alpha) = e^{-\lambda\alpha}$; note that a newly arrived item has a weight of $f(0) = 1$. As discussed later, R-TBS ensures that the probability that an item appears in the sample is proportional to its weight. All items arriving at the same time have the same weight, so that the *total weight* of all items seen up through time t_k is $W_k = \sum_{i=1}^{k} |\mathcal{B}_i| f(\alpha_{i,k})$.

ALGORITHM 2: Generating a sample from a latent sample

```
1  $L = (A, \pi, C)$: latent sample
2  $U \leftarrow$ UNIFORM()
3  if $U \le \mathrm{frac}(C)$ then $S \leftarrow A \cup \pi$ else $S \leftarrow A$
4  return $S$
```

Latent samples: The other key concept for R-TBS is the notion of a *latent sample*, which formalizes the idea of a sample of fractional size. Formally, given a set U of items, a *latent sample* of U with *sample weight* C is a triple $L = (A, \pi, C)$, where $A \subseteq U$ is a set of $\lfloor C \rfloor$ *full* items and $\pi \subseteq U$ is a (possibly empty) set containing at most one *partial* item; π is nonempty if and only if $C > \lfloor C \rfloor$.

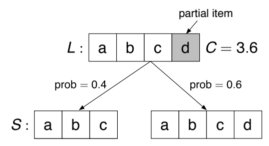


Figure 1: Latent sample L (sample weight $C = 3.6$) and possible realized samples

We randomly generate a sample S from L by sampling as described in Algorithm 2, where $\mathrm{frac}(x) = x - \lfloor x \rfloor$; e.g., see Figure 1. In the pseudocode, the function UNIFORM() generates a random number uniformly distributed on $[0, 1]$. Each full item is included with probability 1 and the partial item is included with probability $\mathrm{frac}(C)$, and it is easy to show that $\mathrm{E}[|S|] = C$. By allowing at most one partial item, we minimize the latent sample's footprint: $|A \cup \pi| \le \lfloor C \rfloor + 1$. Importantly, if the weight C of a latent sample L is an integer, then L contains no partial item, and the sample S generated from L via Algorithm 2 is unique and contains exactly C items.

Downsampling: Besides extracting an actual sample from a latent sample, another key operation on latent samples is *downsampling* (Algorithm 3). For $\theta \in [0, 1]$, the goal of downsampling $L = (A, \pi, C)$ is to obtain an new latent sample $L' = (A', \pi', \theta C)$ such that, if we generate S and S' from C and C' via Algorithm 2, we have

$$\Pr[x \in S'] = \theta \Pr[x \in S] \tag{2}$$

for all $x \in S$. Thus all of the the appearance probabilities, as well as the sample weight (and hence expected sample size), are scaled down by a factor of θ. R-TBS uses downsampling to remove sample items that either decay or are overwritten by arriving items, and also to initially filter the items in an arriving batch.

In the pseudocode, the subroutine SWAP1(A, π) moves a randomly selected item from A to π and moves the current item in π (if any) to A. Similarly, MOVE1(A, π) moves a randomly selected item from A to π, replacing the current item in π (if any).

ALGORITHM 3: Downsampling

```
1  $L = (A, \pi, C)$: input latent sample
2  $\theta$: scaling factor with $\theta \in [0, 1]$
3  if $C = 0$ then return $L' = (\emptyset, \emptyset, 0)$              //sample is empty
4  $U \leftarrow$ UNIFORM(); $C' = \theta C$
5  if $\lfloor C' \rfloor = 0$ then                              //no full items retained
6      if $U > \mathrm{frac}(C)/C$ then
7          $(A', \pi') \leftarrow$ SWAP1$(A, \pi)$
8      $A' \leftarrow \emptyset$
9  else if $0 < \lfloor C' \rfloor = \lfloor C \rfloor$ then                        //no items deleted
10     if $U > \big(1 - \theta\,\mathrm{frac}(C)\big)/\big(1 - \mathrm{frac}(C')\big)$ then
11         $(A', \pi') \leftarrow$ SWAP1$(A, \pi)$
12 else                                   //items deleted: $0 < \lfloor C' \rfloor < \lfloor C \rfloor$
13     if $U \le \theta\,\mathrm{frac}(C)$ then
14         $A' \leftarrow$ SAMPLE$(A, \lfloor C' \rfloor)$
15         $(A', \pi') \leftarrow$ SWAP1$(A', \pi)$
16     else
17         $A' \leftarrow$ SAMPLE$(A, \lfloor C' \rfloor + 1)$
18         $(A', \pi') \leftarrow$ MOVE1$(A', \pi)$
19 if $C' = \lfloor C' \rfloor$ then                              //no fractional item
20     $\pi' \leftarrow \emptyset$
21 return $L' = (A', \pi', C')$
```

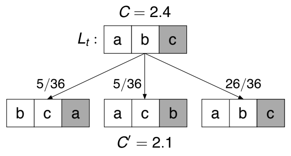


Figure 2: Downsampling example: $C = 2.4 \to C' = 2.1$

To gain some intuition for why the algorithm works, consider a special case (the if-statement in line 9): the goal is to form a latent sample $L' = (A', \pi', \theta C)$ from a latent sample $L = (A, \pi, C)$, where L and L' have the same number of full items and each has one partial item; e.g., $C = 2.4$ and $C' = 2.1$. In this case, the partial item $x^* \in \pi$ either becomes full by being swapped into A' or remains as the partial item for L'. The symmetric treatment of the items in A ensures that appearance probabilities are scaled down uniformly. Denoting by β the probability of *not* swapping, we have $P[x^* \in S'] = \beta \cdot \mathrm{frac}(C') + (1 - \rho) \cdot 1$. On the other hand, (2) implies that $P[x^* \in S'] = \theta\,\mathrm{frac}(C)$. Equating these expressions shows that β must equal the formula on the right side of the inequality on line 10; see Figure 2. The other possible downsampling scenarios are described in [12].

Theorem 2. *For $\theta \in [0, 1]$, let $L' = (A', \pi', \theta C)$ be the latent sample produced from a latent sample $L = (A, \pi, C)$ via Algorithm 3, and let S' and S be samples produced from L' and L via Algorithm 2. Then $\Pr[x \in S'] = \theta \Pr[x \in S]$ for all $x \in A \cup \pi$.*

The union operator: We also need to take the union of disjoint latent samples while preserving the inclusion probabilities for each. Two latent samples $L_1 = (A_1, \pi_1, C_1)$ and $L_2 = (A_2, \pi_2, C_2)$ are *disjoint* if $(A_1 \cup \pi_1) \cap (A_2 \cup \pi_2) = \emptyset$. The pseudocode for the union operation is given as Algorithm 4. The idea is to add all full items to the combined latent sample. If there are partials items in L_1 and L_2, then we transform them to either a single partial item, a full item, or a full plus partial item, depending on the values of $\mathrm{frac}(C_1)$ and $\mathrm{frac}(C_2)$. Such transformations are done in a manner that preserves the appearance probabilities. We obtain the union of multiple latent samples by iterating Algorithm 4; for latent samples $L_1, \ldots, L_k$, we denote the resulting latent sample by $\bigcup_{j=1}^{k} L_j$.

ALGORITHM 4: Union

```
1  $L_1 = (A_1, \pi_1, C_1)$: fractional sample of size $C_1$
2  $L_2 = (A_2, \pi_2, C_2)$: fractional sample of size $C_2$
3  $C \leftarrow C_1 + C_2$
4  $U \leftarrow$ Uniform()
5  if frac($C_1$) + frac($C_2$) < 1 then
6      $A \leftarrow A_1 \cup A_2$
7      if $U \le \mathrm{frac}(C_1)/(\mathrm{frac}(C_1) + \mathrm{frac}(C_2))$ then $\pi \leftarrow \pi_1$ else
         $\pi \leftarrow \pi_2$
8  else if frac($C_1$) + frac($C_2$) = 1 then
9      $\pi \leftarrow \emptyset$
10     if $U \le \mathrm{frac}(C_1)$ then $A \leftarrow A_1 \cup A_2 \cup \pi_1$ else
         $A \leftarrow A_1 \cup A_2 \cup \pi_2$
11 else //frac($C_1$) + frac($C_2$) > 1
12     if $U \le (1 - \mathrm{frac}(C_1)) \,/\, [(1 - \mathrm{frac}(C_1)) + (1 - \mathrm{frac}(C_2))]$ then
13         $\pi = \pi_1$
14         $A \leftarrow A_1 \cup A_2 \cup \pi_2$
15     else
16         $\pi = \pi_2$
17         $A \leftarrow A_1 \cup A_2 \cup \pi_1$
18 return L=(A,$\pi$, C)
```

Theorem 3. *Let $L_1 = (A_1, \pi_1, C_1)$ and $L_2 = (A_2, \pi_2, C_2)$, be disjoint latent samples, and let $L = (A, \pi, C)$ be the latent sample produced from L_1 and L_2 by Algorithm 4. Let S_1, S_2, and S be random samples generated from L_1, and L_2, and L via Algorithm 2. Then (i) $C = C_1 + C_2 = \mathrm{E}[S]$, (ii) $\forall x \in L_1$, $\Pr[x \in S] = \Pr[x \in S_1]$, and (iii) $\forall x \in L_2$, $\Pr[x \in S] = \Pr[x \in S_2]$.*

4.2 The R-TBS Algorithm

The algorithm: R-TBS is given as Algorithm 5. The algorithm generates a sequence of latent samples $\{L_k\}_{k\ge1}$ and from these generates a sequence of actual samples $\{S_k\}_{k\ge1}$ that are returned to the user. In the algorithm, the functions Getsample, Downsample, and Union execute the operations described in Algorithms 2, 3, and 4.

The goal of the algorithm is to ensure that

$$\Pr[x \in S_k] = \rho_k f(\alpha_{i,k}) \tag{3}$$

for all $k \ge 1$, $i \le k$, and $x \in \mathcal{B}_i$, where $f(\alpha) = e^{-\lambda\alpha}$ and $\{\rho_k\}_{k\ge1}$ are the successive values of the variable ρ during a run of the algorithm. Clearly, (3) immediately implies (1). We choose ρ_k to make the sample size as large as possible without exceeding n. Indeed, we show in Theorem 4 below that $C_k = \rho_k W_k$ for all k, and therefore set $\rho_k = \min(1, n/W_k)$, so that $C_k = \min(W_k, n)$. Thus if $W_k < n$, then the sample weight is at its maximum possible value W_k, leading to the maximum possible sample size of $\lfloor W_k \rfloor$ or $\lceil W_k \rceil$. If $W_k \ge n$, then the sample weight, and hence the sample size, is capped at n.

R-TBS functions similarly to classic reservoir sampling. When a new batch of items arrives, all of the items are accepted if the cumulative set of (weighted) items plus the batch items fit in the reservoir of size n ($\rho = 1$ in line 9). If the total item weight exceeds n just before the batch arrives, then a random subset of old items is removed from the sample via downsampling ($\rho/\rho' < 1$ in line 8, over and above decay θ) and a random subset of the arriving items, also filtered via downsampling ($\rho < 1$ in line 9), take their place (line 10). The algorithm also correctly handles the intermediate case where all cumulative items fit, but inserting all arriving items would cause the reservoir to overflow; in this case, only some of the new items overwrite sample items.

ALGORITHM 5: Reservoir-based TBS (R-TBS)

```
1  $\theta = e^{-\lambda\Delta}$: decay factor
2  $n$: maximum sample size
3  Initialize: $W \leftarrow 0$; $A \leftarrow \emptyset$; $\pi \leftarrow \emptyset$; $C \leftarrow 0$; $\rho \leftarrow 1$
4  for $k \leftarrow 1, 2, \ldots$ do
5      $W \leftarrow \theta W + |\mathcal{B}_k|$                              //update total weight
6      $\rho' \leftarrow \rho$
7      $\rho \leftarrow \min(1, n/W)$                                  //update $\rho$
8      $(A, \pi, C) \leftarrow$ Downsample$((A, \pi, C), (\rho/\rho')\theta)$   //decay old items
9      $L_0 \leftarrow$ Downsample$((\mathcal{B}_k, \emptyset, |\mathcal{B}_k|), \rho)$          //take in new items
10     $L \leftarrow$ Union$(L_0, (A, \pi, C))$               //combine old and new items
11     $S \leftarrow$ Getsample$(L)$
12     output $S$
```

Algorithm properties: Theorem 4(i) below asserts that R-TBS satisfies (3) and hence (1), thereby maintaining the correct inclusion probabilities. Theorem 4(ii) implies that the sample size and stability are maximized, as formalized in Theorem 5 below. Finally, the assertion in Theorem 4(iii) ensures that the inclusion probabilities for a given item are nonincreasing over time. This is crucial, since otherwise we might have to recover an item that was previously deleted from the sample, which is impossible.

Theorem 4. *Let $\{L_k = (A_k, \pi_k, C_k)\}_{k\ge1}$ and $\{S_k\}_{k\ge1}$ be a sequence of latent samples and samples, respectively, produced by Algorithm 5 and define $\rho_k = \min(1, n/W_k)$. Then (i) $\Pr[x \in S_k] = \rho_k f(\alpha_{i,k})$ for all $1 \le i \le k$ and $x \in \mathcal{B}_i$, (ii) $C_k = \rho_k W_k$ for all k, and (iii) $\rho_k f(\alpha_{i,k}) \le \rho_{k-1} f(\alpha_{i,k-1})$ for all $1 \le i < k$.*

A sample S_k is *unsaturated* if $C_k < n$ and *saturated* if $C_k = |S_k| = n$; note that $W_k < n$ if and only if S_k is unsaturated. Theorem 5 asserts that, among all sampling schemes with exponential time biasing, R-TBS both maximizes the expected sample size in unsaturated scenarios and minimizes sample-size variability. Thus R-TBS tends to yield more accurate ML results (via more training data) and greater stability in both result quality and retraining costs.

Theorem 5. *Let H be any sampling algorithm for exponential decay that satisfies (1) and denote by S_k and S_k^H the samples produced at (arbitrary) time t_k by R-TBS and H. Then (i) if $W_k < n$, then $\mathrm{E}[|S_k^H|] \le \mathrm{E}[|S_k|]$, and (ii) if $\mathrm{E}[|S_k^H|] = \mathrm{E}[|S_k|]$, then $\mathrm{Var}[|S_k^H|] \ge \mathrm{Var}[|S_k|]$.*

Indeed, (1) implies that, for any $t_i \le t_k$ and $x \in \mathcal{B}_i$, the inclusion probability $\Pr[x \in S_k^H]$ must be of the form $r_k^H f(\alpha_{i,k})$ for some function r_k^H independent of i. Taking $i = k$, we see that $r_k^H \le 1$. For R-TBS with $C_k < n$, Theorem 4 implies that $r_k^H = \rho_k = C_k/W_k = 1$, so that $\Pr[x \in S_k^H] \le \Pr[x \in S_k]$, proving (i). To prove (ii), observe that, over all possible sample-size distributions having mean value equal to $C_k = \mathrm{E}[|S_k|]$, the variance is minimized by concentrating all of the probability mass onto $\lfloor C_k \rfloor$ and $\lceil C_k \rceil$, and this is precisely the sample-size distribution attained by R-TBS.

5. DISTRIBUTED TBS ALGORITHMS

We now describe the distributed implementation of T-TBS and R-TBS, denoted as D-T-TBS and D-R-TBS.

Overview of D-T-TBS: The D-T-TBS implementation is very similar to the simple distributed Bernoulli time-biased sampling algorithm in [15]. It is embarrassingly parallel, requiring no coordination. At each time point t_k, each worker in the cluster subsamples its partition of the sample with probability p, subsamples its partition of $\mathcal{B}_k$ with probability q, and then takes a union of the resulting data sets.

Overview of D-R-TBS: This algorithm, unlike D-T-TBS, maintains a bounded sample, and hence is not embarrassingly parallel. D-R-TBS first needs to aggregate the local partition sizes for the incoming batch $\mathcal{B}_k$ to compute the total batch size $|\mathcal{B}_k|$ and calculate the new total weight W_k. Then, based on $|\mathcal{B}_k|$, W_k, and the current sample weight C_k, D-R-TBS computes the downsample rate for the items in the reservoir, as well as the downsample rate for the items in $\mathcal{B}_k$. After that, D-R-TBS chooses the items in the reservoir to delete through a Downsample operation, selects items in $\mathcal{B}_k$ (also via Downsample), inserts the selected items into the reservoir (via Union), and finally generates the sample (via Getsample). The expensive operations Downsample, Union, and Getsample are all performed in a distributed manner. They each require the master to coordinate among the workers. Getsample and Union operations are relatively straightforward. The most challenging part of D-R-TBS lies in choosing items to delete from the reservoir and selecting new items to insert; we introduce two alternative approaches in Section 5.2. The implementation details for D-T-TBS are mostly subsumed by those for D-R-TBS, so we focus on the latter.

5.1 Distributed Data Structures

There are two important data structures in D-R-TBS: the incoming batch and the reservoir. Conceptually, we view an incoming batch $\mathcal{B}_k$ as an array of slots numbered from 1 through $|\mathcal{B}_k|$, and the reservoir L as an array of slots numbered from 1 through $\lfloor C_k \rfloor$ containing full items plus a special slot for the partial item. For both data structures, data items need to be distributed into partitions due to the large data volumes. Therefore, the slot number of an item, s, maps to a pair (p_s, r_s), where p_s is the partition ID and r_s is the position inside the partition.

Incoming batches usually come from a distributed streaming system, such as Spark Streaming; the actual data structure is specific to the streaming system (e.g. an incoming batch is stored as an RDD in Spark Streaming). As a result, the partitioning strategy of the incoming batch is opaque to D-R-TBS. Unlike the incoming batch, which is read-only and discarded at the end of each time period, the reservoir data structure must be continually updated. An effective strategy for storing and operating on the reservoir is thus crucial for good performance. We now explore alternative approaches to implementing the reservoir.

Distributed in-memory key-value store: One natural approach implements the reservoir using an off-the-shelf distributed in-memory key-value (KV) store, such as Redis [3] or Memcached [2]. Each item in the reservoir is stored as a KV pair, with the slot number as the key and the item as the value. The partial item has a special slot number such as -1. Inserts and deletes to the reservoir naturally translate into put and delete operations to the KV store.

There are three major limitations to this approach. First, the hash-based or range-based data-partitioning scheme used by a distributed KV store yields reservoir partitions that do not correlate with the partitions of incoming batch. When items from a given partition of an incoming batch are inserted into the reservoir, the inserts touch many (if not all) partitions of the reservoir, incurring heavy network I/O. Second, KV stores incur unnecessary concurrency-control overhead. For each batch, D-R-TBS already carefully coordinates the deletes and inserts so that no two delete or insert operations access the same slots in the reservoir and there is no danger of write-write or read-write conflicts. Finally, the KV store approach requires an explicit slot number for each item. As a result, D-R-TBS needs to take extra care to make sure that after deletes and inserts of reservoir items, the slot numbers are still unique and contiguous, e.g. by recycling the slot numbers of deleted items for new inserts. The burden of keeping track of delete and insert slot numbers falls on the master node.

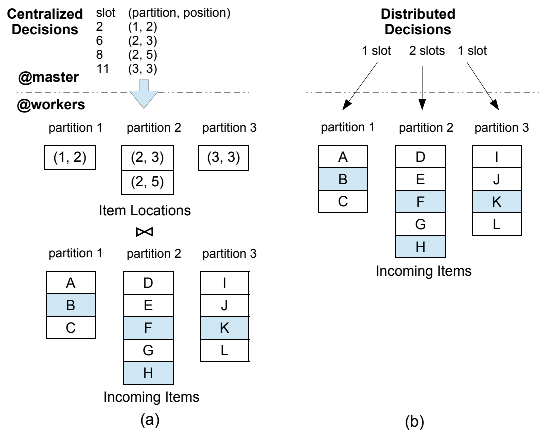


Figure 3: Retrieving insert items

Co-partitioned reservoir: An alternative approach implements a distributed in-memory data structure for the reservoir so as to ensure that the reservoir partitions coincide with the partitions from incoming batches. This can be achieved in spite of the unknown partitioning scheme of the streaming system. Specifically, the reservoir is initially empty, and all items in the reservoir are from the incoming batches. Therefore, if an item from a given partition of an incoming batch is always inserted into the corresponding "local" reservoir partition and deletes are also handled locally, then the co-partition and co-location of the reservoir and incoming batch partitions is automatic. For our experiments, we implemented the co-partitioned reservoir in Spark using the in-place updating technique for RDDs in [15]; see [12].

Note that, with co-partitioned reservoir, the mapping between a specific full item and its current slot number may change over time due to insertions and deletions. This does not cause any statistical issues, because the set-based R-TBS algorithm is oblivious to specific slot numbers.

5.2 Choosing Items to Delete and Insert

To bound the sample size, D-R-TBS must carefully coordinate workers when choosing the items to delete from, and insert into, the reservoir. It must also ensure the statistical correctness of random number generation and random permutation operations in the distributed environment. We consider two possible approaches, focusing on the co-partitioned reservoir; see [12] for the KV store version.

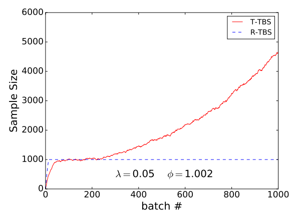
Centralized decisions: In the most straightforward approach, the master makes centralized decisions. For inserts, the master generates the slot numbers of the incoming items $\mathcal{B}_k$ at time t_k that need to be inserted into the reservoir. Suppose that $\mathcal{B}_k$ comprises $m \geq 1$ partitions. Each generated slot number $s \in \{1, 2, \ldots, |\mathcal{B}_k|\}$ is mapped to an item location indicated by (p_s, r_s). Denote by $\mathcal{Q}$ the set of item locations, i.e., the set of (p_s, r_s) pairs. In order to perform the inserts, D-R-TBS needs to first retrieve the actual items based on the item locations. This can be achieved with a join-like operation between $\mathcal{Q}$ and $\mathcal{B}_k$, with the (p_s, r_s) pair matching the actual location of an item inside $\mathcal{B}_k$. To optimize this operation, we make $\mathcal{Q}$ a distributed data structure and use a customized partitioner to ensure that all pairs (p_s, r_s) with $p_s = j$ are co-located with partition j of $\mathcal{B}_k$ for $j = 1, 2, \ldots, m$. Then a co-partitioned and co-located join can be carried out between $\mathcal{Q}$ and $\mathcal{B}_k$, as illustrated in Figure 3(a) for $m = 3$. The resulting set of retrieved insert items, denoted as $\mathcal{S}$, is also co-partitioned with $\mathcal{B}_k$ as a by-product. After that, the actual inserts are carried out depending on the reservoir representation (KV store or co-partitioned reservoir). For the co-partitioned reservoir, we simply use a join-like operation on $\mathcal{S}$ and the reservoir L to add the corresponding insert items to the co-located partition of L. Similarly, for deletes, the master generates slot numbers of the reservoir items to be deleted, then deletes are executed based on the reservoir representation. For the co-partitioned reservoir, we again use a customized partitioner for the set of (p_s, r_s) pairs that represent the slot numbers, denoted as $\mathcal{R}$, such that deletes are co-located with the corresponding L partitions. Then a join-like operation on $\mathcal{R}$ and L performs the actual delete operations on the reservoir.

Distributed decisions: The above approach requires the master to generate large quantities of slot numbers, so we now explore an alternative approach that offloads the slot number generation to the workers while still ensuring the statistical correctness of the computation. This approach has the master choose only the number of deletes and inserts per worker according to an appropriate multivariate hypergeometric distribution. For deletes, each worker chooses random victims from its local partition of the reservoir based on the number of deletes given by the master. For inserts, the worker randomly and uniformly selects items from its local partition of the incoming batch $\mathcal{B}_k$ given the number of inserts. Figure 3(b) depicts how the insert items are retrieved under this decentralized approach. We use the technique in [11] for parallel pseudo-random number generation.

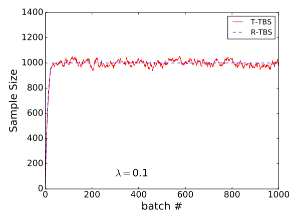
The foregoing distributed decision making approach works only when the co-partitioned reservoir is used. This is because the KV store approach requires a target reservoir slot number for each insert item from the incoming batch, and the target slot numbers have to be generated in such a way as to ensure that, after the deletes and inserts, all of the slot numbers are still unique and contiguous in the new reservoir. This requires a lot of coordination among the workers, which inhibits truly distributed decision making.

6. EXPERIMENTS

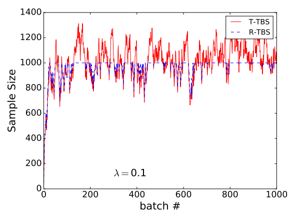
We briefly highlight some of our experimental results; see [12] for details and additional experiments. We implemented R-TBS and T-TBS on Spark. Data was streamed in from HDFS using Spark Streaming's microbatches. All performance experiments were conducted on a cluster of 9 ProLiant DL160 G6 servers interconnected by 1 Gbit Ethernet. Decay occurs according to a time scale such that the batch-arrival interval is $\Delta = 1$ in the decay formulas.



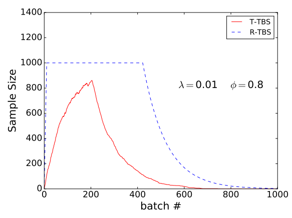
(a) Growing Batch Sz.



(b) Stable Batch Sz. (Det.)



(c) Stable Batch Sz. (Unif.)



(d) Decaying Batch Sz.

Figure 4: Sample size behavior of T-TBS and R-TBS

Sample size Behavior: Figures 4 shows sample size behavior of T-TBS and R-TBS under a variety of batch-size regimes. In Figure 4(a), the (deterministic) batch size is initially fixed and the algorithm is tuned to a target sample size of 1000, with a decay rate of $\lambda = 0.05$. At $k = 200$, the batch size starts to increase: $B_{k+1} = \phi B_k$, where $\phi = 1.002$. This leads to an overflowing sample for T-TBS, whereas R-TBS maintains a constant sample size. Even in a stable batch-size regime with batch sizes either constant (Figure 4(b); $B_k \equiv 100$ with $\lambda = 0.1$) or fluctuating (Figure 4(c); B_k uniform on $[0, 200]$), R-TBS can maintain a bounded sample size, whereas the sample size under T-TBS fluctuates per Theorem 1; as in Theorem 5, the R-TBS unsaturated sample size is always larger than than for T-TBS. For $\phi = 0.8$, so that the batch sizes start to shrink at $k = 200$, Figure 4(d) shows that R-TBS is more robust to sample underflows.

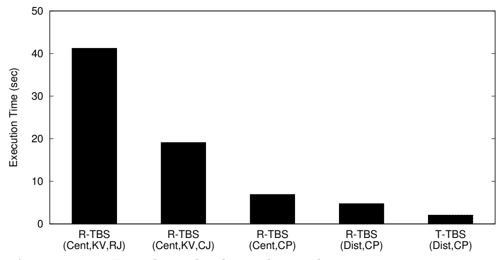


Figure 5: Per-batch distributed runtime comparison

Runtime performance: Figure 5 shows the average runtime per batch for five different implementations of distributed TBS algorithms. Each batch contains 10 million items. The first four are D-R-TBS implementations with different design choices: whether to use centralized or distributed decisions in choosing items to insert and delete (abbreviated as "Cent" and "Dist", respectively), whether to implement the reservoir using a key-value store or a co-partitioned reservoir scheme (abbreviated as "KV" and "CP"), and whether to subsample the incoming batch using the standard repartition join or using a copartitioned join (abbreviated as "RJ" and "CJ") under the centralized decision scheme. As can be seen, the best implementation is almost an order of magnitude faster than the worst. Since D-T-TBS is embarrassingly parallelizable, it is much faster than the best D-R-TBS implementation (see rightmost bar). But, as discussed in Section 3, T-TBS only works under very strong restrictions on the data arrival process, and can suffer from occasional memory overflows.

We have also conducted scalability experiments and evaluated the impact of the decay factor as well as batch-size skew on the runtime performance; see [12]. With 8 workers, our implementation of R-TBS can handle 100 million items arriving approximately every 16 seconds.

Application to ML models: We first compare the performance of R-TBS, simple sliding windows (SW), and uniform sampling (Unif) when applied to a kNN classifier that predicts a class for each item in an incoming batch and then updates the sample. We use 100 classes, and the data generation process operates in one of two “modes”. In the “normal” mode, the frequency of items from any of the first 50 classes is five times higher than that of items in any of the second 50 classes. In the “abnormal” mode, the frequencies are five times lower. The sample size for both R-TBS and Unif is 1000, and SW contains the last 1000 items; thus all methods use the same amount of data for retraining.

Figure 6 shows the misclassification rates for the three sampling methods under a periodic pattern of 10 normal batches alternating with 10 abnormal batches, denoted as P(10, 10). When the data distribution first becomes abnormal at $t = 10$, the misclassification rates under all sampling schemes increase sharply. R-TBS and SW adapt to the new mode, with SW adapting slightly faster. (Unif never adapts at all.) After the first mode change, however, R-TBS “remembers” both normal and abnormal values, and thereby becomes much more robust to subsequent mode changes, whereas SW continues to overreact with wild fluctuations.

Table 1 displays both the accuracy and robustness of Unif, SW, and R-TBS (using several values of λ) over 30 runs. Accuracy is measured in terms of the average misclassification rate, and robustness is measured as the average 10% *expected shortfall (ES)* , i.e., the average value of the worst 10% of cases [14, p. 70]. Results are shown for a set of temporal patterns that include several periodic patterns and a “single event” comprising one normal-abnormal-normal cycle. As can be seen, R-TBS and SW have similar accuracies, and Unif is always the worst by a large margin. R-TBS is always best in terms of robustness and SW is always the worst, with ES values 1.5 to 2.5 times higher than for R-TBS. Unif also does poorly in terms of robustness, except for the single event, since the data remains in normal mode after the abnormal period and time biasing becomes unimportant. Overall, R-TBS provides superior accuracy and robustness, and this performance edge is fairly stable across a wide range of λ values.

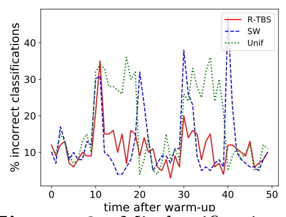


Figure 6: Misclassification rate (percent) for kNN: n=1000, P(10, 10)

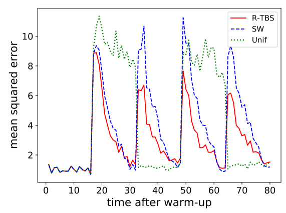


Figure 7: Mean square error for linear regression: n=1600, P(16, 16)

Table 1: Accuracy and robustness of kNN performance

	Single Event		P(10,10)		P(20,10)		P(30,10)	
λ	Miss%	ES	Miss%	ES	Miss%	ES	Miss%	ES
0.05	17.1	**16.8**	16.1	22.1	15.3	24.4	15.1	25.9
0.07	16.5	17.3	15.3	**21.3**	14.9	**24.0**	**14.4**	**25.2**
0.10	**15.7**	18.5	**15.1**	22.1	**14.7**	24.9	14.7	26.9
SW	19.2	42.1	17.1	41.7	16.1	39.8	15.9	38.3
Unif	21.3	18.3	25.4	34.8	19.6	35.7	19.0	35.8

Figure 7 shows similar results for an experiment involving a regression model. Interestingly, the parameters were such that the R-TBS sample was never full, whereas SW and Unif were always full. This shows that a smaller sample with good ratios of old and new data can provide better prediction performance than a larger but temporally unbalanced sample.

7. CONCLUSION & FUTURE WORK

Our experiments with ML models and graph analytics [15], indicate the usefulness of periodic retraining over time-biased samples to help ML algorithms robustly deal with evolving data streams without requiring algorithmic re-engineering.

In ongoing work [12], we have extended our R-TBS and T-TBS sampling schemes to arbitrary decay functions. Theory and algorithms are more complex in this setting because, unlike the exponential case, decay factors now vary by age, so item ages must be tracked. R-TBS then satisfies (1) only approximately, with an error that can be made arbitrarily small by increasing the sample footprint. There is also a well defined trade-off between sample footprint and sample-size stability and saturation. Interesting future directions are to apply our ideas to other types of streaming analytics, and to develop end-to-end solutions via drift-detection techniques.

8. REFERENCES

[1] An interactive deep dive into the Kaggle data science survey. https://www.kaggle.com/sudalairajkumar/an-interactive-deep-dive-into-survey-results.

[2] Memcached. https://memcached.org.

[3] Redis. https://redis.io.

[4] C. C. Aggarwal. On biased reservoir sampling in the presence of stream evolution. In *VLDB*, pages 607–618. VLDB Endowment, 2006.

[5] P. Bailis, E. Gan, S. Madden, D. Narayanan, K. Rong, and S. Suri. MacroBase: Prioritizing attention in fast data. In *SIGMOD*, pages 541–556, 2017.

[6] M. T. Chao. A general purpose unequal probability sampling plan. *Biometrika*, pages 653–656, 1982.

[7] E. Cohen and M. J. Strauss. Maintaining time-decaying stream aggregates. *J. Algo.*, 59(1):19–36, 2006.

[8] P. S. Efraimidis and P. G. Spirakis. Weighted random sampling with a reservoir. *Inf. Process. Lett.*, 97(5):181–185, 2006.

[9] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4):44, 2014.

[10] R. Gemulla and W. Lehner. Sampling time-based sliding windows in bounded space. In *SIGMOD*, pages 379–392, 2008.

[11] H. Haramoto, M. Matsumoto, T. Nishimura, F. Panneton, and P. L’Ecuyer. Efficient Jump Ahead for 2-Linear Random Number Generators. *INFORMS Journal on Computing*, 20(3):385–390, 2008.

[12] B. Hentschel, P. J. Haas, and Y. Tian. Temporally-biased sampling schemes for online model management. *CoRR*, abs/1906.05677, 2019.

[13] V. Kachitvichyanukul and B. W. Schmeiser. Binomial random variate generation. *Commun. ACM*, 31(2):216–222, 1988.

[14] A. J. McNeil, R. Frey, and P. Embrechts. *Quantitative Risk Management: Concepts, Techniques and Tools*. Second edition, 2015.

[15] W. Xie, Y. Tian, Y. Sismanis, A. Balmin, and P. J. Haas. Dynamic interaction graphs with probabilistic edge decay. In *ICDE*, pages 1143–1154, 2015.

Technical Perspective: Succinct Range Filters

Stratos Idreos
Harvard University

Data structures that filter data for point or range queries are prevalent across all data-driven applications, from analytics to transactions, and modern machine learning applications. The primary objective is simple: find whether one or more data items exist in the database. Yet, this simple task is exceptionally hard to perform efficiently, and surprisingly critical for the overall properties of the data-intensive applications that rely on filtering.

This is a hard problem as there are numerous critical parameters and trade-offs. Many parameters come from the workload, e.g., the exact percentage of point queries versus updates, percentage of empty-result queries, etc. Other parameters come from the underlying hardware; e.g., filters typically reside in memory but, with exponentially increasing data sizes, we need to be mindful of the filter size and the memory hierarchy. Overall, there are complex trade-offs to navigate: memory, read, and write amplification. For example, a data structure cannot be efficient for both point and range queries while also supporting efficient writes. Yet, numerous applications need to expose both read patterns.

A prototypical application of filters is LSM-tree storage engines. An LSM-tree stores data in the order they arrive in immutable files and periodically sort-merges them into larger files. This way, it behaves in between a log and a sorted array, providing a good balance of read and write performance depending on the exact tuning (file size, buffer size, etc.). LSM-tree storage engines are used as the backbone of most distributed key-value stores and applications range from social media, web-applications, e-shopping, IoT, etc. Due to their multi-level architecture enforcing a global temporal order, LSM-tree engines rely heavily on in-memory filters.

In ACM SIGMOD 2018, Succinct Range Filters (SuRF) was introduced as a new succinct filter that can handle point queries, range queries, and approximate counts efficiently [1]. SuRF is based on a trie-like structure termed Fast Succinct Trie. The trie-based design allows building a structure that can support performant range queries and point queries.

The authors of SuRF make the following critical and insightful observation which brings everything together and allows SuRF to balance the various hardware and workload trade-offs. For a given set of queries, the upper levels of the trie incur many more accesses than the lower levels. For this reason, the SuRF design utilizes a dense, performance-optimized encoding scheme for the top of the trie and a sparse, memory-optimized encoding scheme for the bottom. This results in a data structure that is both fast and memory efficient. The upper levels, which are comprised of few nodes but incur many accesses, encode keys under the LOUDS-Dense scheme, sacrificing space efficiency for fast lookups. The lower levels, which contain the majority of nodes but have a sparse access pattern relative to high levels are encoded with LOUDS-Sparse, sacrificing fast lookups for space efficiency.

Compared to state-of-the-art bloom filter based solutions (e.g., prefix bloom filters) SuRF provides a general solution, i.e., it can support any range query as well as efficient point queries. Compared to state-of-the-art tree or trie based solutions SuRF offers similar or better performance at a much smaller memory footprint. The SuRF paper shows end-to-end impact by integrating SuRF in RocksDB, the most mature LSM-tree based storage engine, and demonstrating strong results (e.g., up to 5x) in time-series applications for both point and range queries. SuRF can be applied broadly to any application that needs a succinct filter such as monitoring, privacy/security, graph analytics, etc. Finally, the core spirit of the design of SuRF exemplifies elegant research taste in pursuing hybrid, hardware- and workload-conscious designs.

1. REFERENCES

[1] H. Zhang, H. Lim, V. Leis, D. G. Andersen, M. Kaminsky, K. Keeton, and A. Pavlo. SuRF: Practical Range Query Filtering with Fast Succinct Tries. In ACM SIGMOD 2018.

Succinct Range Filters

Huanchen Zhang[฿], Hyeontaek Lim[฿], Viktor Leis[€], David G. Andersen[฿],
Michael Kaminsky[$], Kimberly Keeton[£], Andrew Pavlo[฿]
[฿]Carnegie Mellon University, [€]Friedrich Schiller University Jena, [$]Intel Labs, [£]Hewlett Packard Labs
huanche1, hl, dga, pavlo@cs.cmu.edu, viktor.leis@uni-jena.de, michael.e.kaminsky@intel.com,
kimberly.keeton@hpe.com

ABSTRACT

We present the *Succinct Range Filter* (SuRF), a fast and compact data structure for approximate membership tests. Unlike traditional Bloom filters, SuRF supports both single-key lookups and common range queries. SuRF is based on a new data structure called the *Fast Succinct Trie (FST)* that matches the point and range query performance of state-of-the-art order-preserving indexes, while consuming only 10 bits per trie node. The false positive rates in SuRF for both point and range queries are tunable to satisfy different application needs. We evaluate SuRF in RocksDB as a replacement for its Bloom filters to reduce I/O by filtering requests before they access on-disk data structures. Our experiments on a 100 GB dataset show that replacing RocksDB's Bloom filters with SuRFs speeds up open-seek (without upper-bound) and closed-seek (with upper-bound) queries by up to 1.5× and 5× with a modest cost on the worst-case (all-missing) point query throughput due to slightly higher false positive rate.

1. INTRODUCTION

Write-optimized log-structured merge (LSM) trees [30] are popular low-level storage engines for general-purpose databases that provide fast writes [1, 34] and ingest-abundant DBMSs such as time-series databases [4, 32]. One of their main challenges for fast query processing is that items could reside in immutable files (SSTables) from all levels [3, 25]. Item retrieval in an LSM tree-based design may therefore incur multiple expensive disk I/Os [30, 34]. This challenge calls for in-memory data structures that can help locate query items. *Bloom filters* are a good match for this task [32, 34] because they are small enough to reside in memory, and they have only "one-sided" errors—if the key is present, then the Bloom filter returns true; if the key is absent, then the filter will likely return false, but might incur a false positive.

Although Bloom filters are useful for single-key lookups ("Is key 42 in the SSTable?"), they cannot handle range queries ("Are there keys between 42 and 1000 in the SSTable?"). With only Bloom filters, an LSM tree-based storage engine must read additional table blocks from disk for range queries. Alternatively, one could maintain an auxiliary index, such as a B+Tree, to support such range queries. The I/O cost of range queries is high enough that LSM tree-based designs often use *prefix Bloom filters* to optimize certain fixed-prefix queries (e.g., "where email starts with com.foo@") [2, 20, 32], despite their inflexibility for more general range queries. The designers of RocksDB [2] have expressed a desire to have a more flexible data structure for this purpose [19]. A handful of approximate data structures, including the prefix Bloom filter, exist that accelerate specific categories of range queries, but none is general purpose.

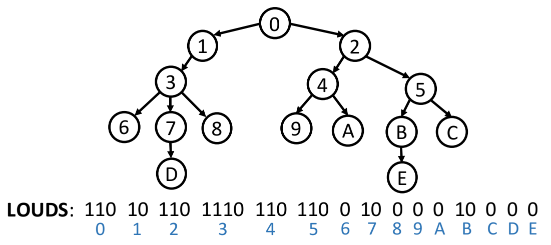
This paper presents the **<u>Su</u>ccinct <u>R</u>ange <u>F</u>ilter** (SuRF), a fast and compact filter that provides exact-match filtering and range filtering. Like Bloom filters, SuRF guarantees one-sided errors for point and range membership tests. SuRF can trade between false positive rate and memory consumption, and this trade-off is tunable for point and range queries semi-independently. SuRF is built upon a new space-efficient (succinct) data structure called the *Fast Succinct Trie (FST)*. It performs comparably to or better than state-of-the-art uncompressed index structures (e.g., B+tree [14], ART [26]) for both integer and string workloads. FST consumes only 10 bits per trie node, which is close to the information-theoretic lower bound.

The key insight in SuRF is to transform the FST into an approximate (range) membership filter by removing levels of the trie and replacing them with some number of suffix bits. The number of such bits (either from the key itself or from a hash of the key—as we discuss later in the paper) trades space for decreased false positives.

We evaluate SuRF via micro-benchmarks and as a Bloom filter replacement in RocksDB. Our experiments on a 100 GB time-series dataset show that replacing the Bloom filters with SuRFs of the same filter size reduces I/O. This speeds up open-range queries (without upper-bound) by 1.5× and closed-range queries (with upper-bound) by up to 5× compared to the original implementation. For point queries, the worst-case workload is when none of the query keys exist in the dataset. In this case, RocksDB is up to 40% slower using SuRFs instead of Bloom filters because they have higher (0.2% vs. 0.1%) false positive rates. One can eliminate this performance gap by increasing the size of SuRFs by a few bits per key.

This paper makes three primary contributions. First, we describe in Section 2 our FST data structure whose space consumption is close to the minimum number of bits required by information theory yet has performance equivalent to uncompressed order-preserving indexes. Second, in Section 3 we describe how to use the FST to build SuRF, an approximate membership test that supports both single-key and range queries. Finally, we replace the Bloom filters with size-matching SuRFs in RocksDB and show that it improves range query performance with a modest cost on the worst-case point query throughput due to a slightly higher false positive rate.

©ACM 2019 This is a minor revision of the paper entitled "SuRF: Practical Range Query Filtering with Fast Succinct Tries", published in SIGMOD'18, ISBN 978-1-4503-4703-7/18/06, June 10–15, 2018, Houston, TX, USA. DOI: https://doi.org/10.1145/3183713.3196931



LOUDS: 110 10 110 1110 110 110 0 10 0 0 0 10 0 0 0
0 1 2 3 4 5 6 7 8 9 A B C D E

Figure 1: An example ordinal tree encoded using LOUDS

2. FAST SUCCINCT TRIES

The core data structure in SuRF is the FST. It is a space-efficient, static trie that answers point and range queries. FST is 4–15× faster than earlier succinct tries using other tree representations [12, 13, 23, 24, 27, 28, 33], achieving performance comparable to or better than the state-of-the-art pointer-based indexes.

FST's design is based on the observation that the upper levels of a trie comprise few nodes but incur many accesses. The lower levels comprise the majority of nodes, but are relatively "colder". We therefore encode the upper levels using a fast bitmap-based encoding scheme (**LOUDS-Dense**) in which a child node search requires only one array lookup, choosing performance over space. We encode the lower levels using the space-efficient **LOUDS-Sparse** scheme, so that the overall size of the encoded trie is bounded.

For the rest of the section, we assume that the trie maps the keys to fixed-length values. We also assume that the trie has a fanout of 256 (i.e., one byte per level).

2.1 Background

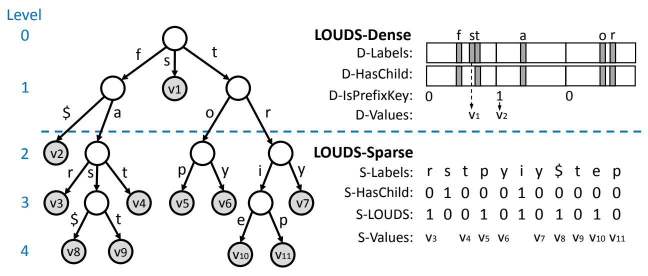
A tree representation is "succinct" if the space taken by the representation is close to the information-theoretic lower bound, which is the minimum number of bits needed to distinguish any object in a class. A class of size n requires at least $\log_2 n$ bits to encode each object. A trie of degree k is a rooted tree where each node can have at most k children with unique labels selected from set $\{0, 1, \ldots, k-1\}$. The information-theoretic lower bound of a trie of degree k is approximately $n(k \log_2 k - (k-1)\log_2(k-1))$ bits [13].

Jacobson [24] pioneered research on succinct tree representations and introduced the ***L****evel-****O****rdered* ***U****nary* ***D****egree* ***S****equence* (LOUDS) to encode an ordinal tree (i.e., a rooted tree where each node can have an arbitrary number of children in order). LOUDS traverses the nodes in a breadth-first order and encodes each node's degree using the unary code. For example, node 3 in Fig. 1 has three children and is thus encoded as `'1110'`. Navigating a tree encoded with LOUDS uses the rank & select primitives. Given a bit vector, $rank_1(i)$ counts the number of 1's up to position i ($rank_0(i)$ counts 0's), while $select_1(i)$ returns the position of the i-th 1 ($select_0(i)$ selects 0's). Modern rank & select implementations [22, 29, 36, 40] achieve constant time by using look-up tables (LUTs) to store a sampling of precomputed results so that they only need to count between the samples.

2.2 LOUDS-Dense

LOUDS-Dense encodes each trie node using three bitmaps of size 256 (because the node fanout is 256) and a byte-sequence for the values as shown in the top half of Fig. 2. The encoding follows level-order (i.e., breadth-first order).

There are three ways to define "close" [9]. Suppose the information-theoretic lower bound is L bits. A representation that uses L+$O(1)$, L+$o(L)$, and $O(L)$ bits is called *implicit*, *succinct*, and *compact*, respectively. All are considered succinct, in general.



Level
0
1
2
3
4
f
s
t
v1
$
a
o
r
v2
r
s
t
p
y
i
y
v3
$
t
v4
v5
v6
e
p
v7
v8
v9
v10
v11
LOUDS-Dense
f st a o r
D-Labels:
D-HasChild:
D-IsPrefixKey: 0 1 0
D-Values: v1 v2
LOUDS-Sparse
S-Labels: r s t p y i y $ t e p
S-HasChild: 0 1 0 0 0 1 0 0 0 0 0
S-LOUDS: 1 0 0 1 0 1 0 1 0 1 0
S-Values: v3 v4 v5 v6 v7 v8 v9 v10 v11
Keys stored: f, far, fas, fast, fat, s, top, toy, trie, trip, try

Figure 2: **LOUDS-DS Encoded Trie** – "`$`" represents the character whose ASCII number is `0xFF`. It is used to indicate the situation where a prefix string leading to a node is also a valid key.

The first bitmap (*D-Labels*) records the branching labels for each node. Specifically, the i-th bit in the bitmap, where $0 \leq i \leq 255$, indicates whether the node has a branch with label i. For example, the root node in Fig. 2 has three outgoing branches labeled **f**, **s**, and **t**. The *D-Labels* bitmap sets the 102nd (**f**), 115th (**s**) and 116th (**t**) bits and clears the rest.

The second bitmap (*D-HasChild*) indicates whether a branch points to a sub-trie or terminates (i.e., points to the value or the branch does not exist). Taking the root node in Fig. 2 as an example, the **f** and the **t** branches continue with sub-tries while the **s** branch terminates with a value. In this case, the *D-HasChild* bitmap only sets the 102nd (**f**) and 116th (**t**) bits for the node.

The third bitmap (*D-IsPrefixKey*) includes only one bit per node. The bit indicates whether the prefix that leads to the node is also a valid key. For example, in Fig. 2, the first node at level 1 has **f** as its prefix. Meanwhile, `'f'` is also a key stored in the trie. To denote this situation, the *D-IsPrefixKey* bit for this child node must be set.

The final byte-sequence (*D-Values*) stores the fixed-length values (e.g., pointers) mapped by the keys. The values are concatenated in level order: same as the three bitmaps.

Tree navigation uses array lookups and rank & select operations. We denote $rank_1$/$select_1$ over bit sequence *bs* on position *pos* to be $rank_1$/$select_1$(*bs*, *pos*). Let *pos* be the current bit position in *D-Labels*. To traverse down the trie, given *pos* where *D-HasChild*[*pos*] = 1, **D-ChildNodePos**(*pos*) = 256 ×$rank_1$(*D-HasChild*, *pos*) computes the bit position of the first child node. To move up the trie, **D-ParentNodePos**(*pos*) = $select_1$(*D-HasChild*, $\lfloor$*pos*/256$\rfloor$) computes the bit position of the parent node. To access values, given *pos* where *D-HasChild*[*pos*] = 0, **D-ValuePos**(*pos*) = $rank_1$(*D-Labels*, *pos*) - $rank_1$(*D-HasChild*, *pos*) + $rank_1$(*D-IsPrefixKey*, $\lfloor$*pos/256*$\rfloor$)-1 gives the lookup position.

2.3 LOUDS-Sparse

As shown in the lower half of Fig. 2, LOUDS-Sparse encodes a trie node using four byte or bit-sequences. The encoded nodes are then concatenated in level-order.

The first byte-sequence (*S-Labels*) records all the branching labels for each trie node. As an example, the first non-value node at level 2 in Fig. 2 has three branches. *S-Labels* includes their labels **r**, **s**, and **t** in order. We denote the case where the prefix leading to a node is also a valid key using the special byte `0xFF` at the beginning of the node (this case is handled by *D-IsPrefixKey* in LOUDS-Dense). For example, in Fig. 2, the first non-value node at level 3 has `'fas'` as its incoming prefix. Since `'fas'` itself is also a stored key, the node adds `0xFF` to *S-Labels* as the first byte. Because the special byte always appears at the beginning of a node, it can be distinguished from the real `0xFF` label.

The second bit-sequence (*S-HasChild*) includes one bit for each byte in *S-Labels* to indicate whether a child branch continues (i.e., points to a sub-trie) or terminates (i.e., points to a value). Taking the rightmost node at level 2 in Fig. 2 as an example, because the branch labeled **i** points to a sub-trie, the corresponding bit in *S-HasChild* is set. The branch labeled **y**, on the other hand, points to a value. Its *S-HasChild* bit is cleared.

The third bit-sequence (*S-LOUDS*) also includes one bit for each byte in *S-Labels*. *S-LOUDS* denotes node boundaries: if a label is the first in a node, its *S-LOUDS* bit is set. Otherwise, the bit is cleared. For example, in Fig. 2, the first non-value node at level 2 has three branches and is encoded as `100` in *S-LOUDS*.

The final byte-sequence (*S-Values*) is organized the same way as *D-Values* in LOUDS-Dense.

Tree navigation on LOUDS-Sparse is as follows: to move down the trie, **S-ChildNodePos**(*pos*) = $select_1$(*S-LOUDS*, $rank_1$(*S-HasChild*, *pos*) + 1); to move up, **S-ParentNodePos**(*pos*) = $select_1$(*S-HasChild*, $rank_1$(*S-LOUDS*, *pos*) - 1); to access a value, **S-ValuePos** (*pos*) = *pos* - $rank_1$(*S-HasChild*, *pos*).

2.4 LOUDS-DS and Operations

LOUDS-DS is a hybrid trie in which the upper levels are encoded with LOUDS-Dense and the lower levels with LOUDS-Sparse. The dividing point between the upper and lower levels is tunable to trade performance and space. FST keeps the number of upper levels small in favor of the space efficiency provided by LOUDS-Sparse. We use a size ratio $1 : R$ between LOUDS-Dense and LOUDS-Sparse to determine the dividing point among levels. Reducing R leads to more LOUDS-Dense levels, favoring performance over space. We use R=64 as the default.

LOUDS-DS supports three basic operations efficiently:

- **ExactKeySearch**(*key*): Return the value of *key* if *key* exists (or NULL otherwise).
- **LowerBound**(*key*): Return an iterator pointing to the key-value pair (k, v) where k is the smallest in lexicographical order satisfying $k \geq key$.
- **MoveToNext**(*iter*): Move the iterator to the next key.

A point query on LOUDS-DS works by first searching the LOUDS-Dense levels. If the search does not terminate, it continues into the LOUDS-Sparse levels. The high-level searching steps at each level are similar regardless of the encoding mechanism: First, search the current node's range in the label sequence for the target key byte. If the key byte does not exist, terminate and return *NULL*. Otherwise, check the corresponding bit in the *HasChild* bit-sequence. If the bit is `1` (i.e., the branch points to a child node), compute the child node's starting position in the label sequence and continue to the next level. Otherwise, return the corresponding value in the value sequence. We precompute two aggregate values based on the LOUDS-Dense levels: the node count and the number of *HasChild* bits set. Using these two values, LOUDS-Sparse can operate as if the entire trie is encoded with LOUDS-Sparse.

Range queries use a high-level algorithm similar to the point query implementation. When performing LowerBound, instead of doing an exact search in the label sequence, the algorithm searches for the smallest label $\geq$ the target label. When moving to the next key, the cursor starts at the current leaf label position and moves forward. If another valid label **l** is found within the node, the algorithm finds the left-most leaf key in the subtree rooted at **l**. If the cursor hits node boundary instead, the algorithm moves the cursor up to the corresponding position in the parent node.

We include per-level cursors in the iterator to minimize the relatively expensive "move-to-parent" and "move-to-child" calls, which require rank & select operations. These cursors record a trace from root to leaf (i.e., the per-level positions in the label sequence) for the current key. Because of the level-order layout of LOUDS-DS, each level-cursor only moves sequentially without skipping items. With this property, range queries in LOUDS-DS are implemented efficiently. Each level-cursor is initialized once through a "move-to-child" call from its upper-level cursor. After that, range query operations at this level only involve cursor movement, which is cache-friendly and fast. Section 4 shows that range queries in FST are even faster than pointer-based tries.

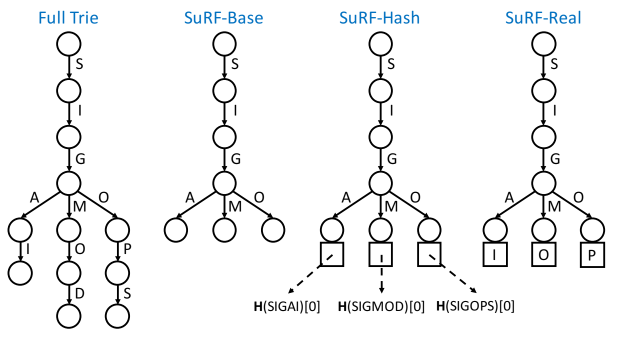


Figure 3: An example of deriving SuRF variations from a full trie.

LOUDS-DS can be built using one scan over a key-value list.

2.5 Space Analysis

Given an n-node trie, LOUDS-Sparse uses $8n$ bits for *S-Labels*, n bits for *S-HasChild* and n bits for *S-LOUDS*, a total of $10n$ bits (plus auxiliary bits for rank & select). Referring to Section 2.1, the information-theoretic lower bound (Z) is approximately $9.44n$ bits. Although the space taken by LOUDS-Sparse is close to the information-theoretic bound, technically, LOUDS-Sparse can only be categorized as *compact* rather than *succinct* in a finer classification scheme because LOUDS-Sparse takes $O(Z)$ space (despite the small multiplier) instead of $Z + o(Z)$.

LOUDS-Dense's size is restricted by the ratio R to ensure that it does not affect the overall space-efficiency of LOUDS-DS. Notably, LOUDS-Dense does not always take more space than LOUDS-Sparse: if a node's fanout is larger than 51, it takes fewer bits to encode the node using the former instead of the latter. Since such nodes are common in a trie's upper levels, adding LOUDS-Dense on top of LOUDS-Sparse often improves space-efficiency.

3. SUCCINCT RANGE FILTERS

In building SuRF using FST, our goal was to balance a low false positive rate with the memory required by the filter. The key idea is to use a truncated trie; that is, to remove lower levels of the trie and replace them with suffix bits extracted from the key (either the key itself or a hash of the key). We introduce four variations of SuRF. We describe their properties and how they guarantee one-sided errors. The current SuRF design is static, requiring a full rebuild to insert new keys.

3.1 Basic SuRF

FST is a trie-based index structure that stores complete keys. As a filter, FST is 100% accurate; the downside, however, is that the full structure can be big. In many applications, filters must fit in memory to guard access to a data structure stored on slower storage. These applications cannot afford the space for complete keys, and thus must trade accuracy for space.

The basic version of SuRF (SuRF-Base) stores the minimum-length key prefixes such that it can uniquely identify each key. Specifically, SuRF-Base only stores an additional byte for each key beyond the shared prefixes. Fig. 3 shows an example. Instead of storing the full keys (`'SIGAI'`, `'SIGMOD'`, `'SIGOPS'`), SuRF-Base truncates the full trie by including only the shared prefix (`'SIG'`) and one more byte for each key (`'C'`, `'M'`, `'O'`).

Pruning the trie in this way affects both filter space and accuracy. Unlike Bloom filters where the keys are hashed, the trie shape of SuRF-Base depends on the distribution of the stored keys. Hence, there is no theoretical upper-bound of the size of SuRF-Base. Empirically, however, SuRF-Base uses only 10 bits per key (BPK) for 64-bit random integers and 14 BPK for emails, as shown in Section 4.2. The intuition is that the trie built by SuRF-Base usually has an average fanout $F > 2$: there are less than twice as many nodes as keys. Because FST (LOUDS-Sparse to be precise) uses 10 bits to encode a trie node, the size of SuRF-Base is less than 20 BPK for $F > 2$.

Filter accuracy is measured by the false positive rate (FPR), defined as $\frac{FP}{FP+TN}$, where *FP* is the number of false positives and *TN* is the number of true negatives. A false positive in SuRF-Base occurs when the prefix of the non-existent query key coincides with a stored key prefix. For example, in Fig. 3, querying key `'SIGMETRICS'` will cause a false positive in SuRF-Base. FPR in SuRF-Base depends on the distributions of the stored and query keys. Our results in Section 4.2 show that SuRF-Base incurs a 4% FPR for integer keys and a 25% FPR for email keys. To improve FPR, we include two forms of key suffixes described below to allow SuRF to better distinguish between the stored key prefixes.

3.2 SuRF with Hashed Key Suffixes

As shown in Fig. 3, SuRF with hashed key suffixes (SuRF-Hash) adds a few hash bits per key to SuRF-Base to reduce its FPR. Let H be the hash function. For each key K, SuRF-Hash stores the n (n is fixed) least-significant bits of $H(K)$ in FST's value array (which is empty in SuRF-Base). When a key (K') lookup reaches a leaf node, SuRF-Hash extracts the n least-significant bits of $H(K')$ and performs an equality check against the stored hash bits associated with the leaf node. Using n hash bits per key guarantees that the point query FPR of SuRF-Hash is less than 2^{-n} (the partial hash collision probability). Even if the point query FPR of SuRF-Base is 100%, just 7 hash bits per key in SuRF-Hash provide a $\frac{1}{2^7} \simeq 1\%$ point query FPR. Experiments in Section 4.2.1 show that SuRF-Hash requires only 2–4 hash bits to reach 1% FPR.

The extra bits in SuRF-Hash do not help range queries because they do not provide ordering information on keys.

3.3 SuRF with Real Key Suffixes

Instead of hash bits, SuRF with real key suffixes (SuRF-Real) stores the n key bits immediately following the stored prefix of a key. Fig. 3 shows an example when $n = 8$. SuRF-Real includes the next character for each key (`'I'`, `'O'`, `'P'`) to improve the distinguishability of the keys: for example, querying `'SIGMETRICS'` no longer causes a false positive. Unlike in SuRF-Hash, both point and range queries benefit from the real suffix bits to reduce false positives. For point queries, the real suffix bits are used the same way as the hashed suffix bits. For range queries (e.g., move to the next key $> K$), when reaching a leaf node, SuRF-Real compares the stored suffix bits s to key bits k_s of the query key at the corresponding position. If $k_s \leq s$, the iterator points to the current key; otherwise, it advances to the next key in the trie.

Although SuRF-Real improves FPR for both point and range queries, the trade-off is that using real keys for suffix bits cannot provide as good FPR as using hashed bits because the distribution correlation between the stored keys and the query keys weakens the distinguishability of the real suffix bits.

3.4 Operations

We summarize how to implement SuRF's basic operations using FST. The key is to guarantee one-sided error (no false negatives).

build*(keyList)*: Construct the filter given a list of keys.

result = ***lookup****(k)*: Perform a point query on k. Returns true if k may exist (could be false positive); false guarantees non-existence. This operation first searches for k in the FST. If the search terminates without reaching a leaf, return false. If the search reaches a leaf, return true in SuRF-Base. In other SuRF variants, fetch the stored key suffix k_s of the leaf node and perform an equality check against the suffix bits extracted from k according to the suffix type as described in Sections 3.2 and 3.3.

iter, fp_flag = ***moveToNext****(k)*: Return an iterator pointing to the smallest key that is $\geq k$. The iterator supports retrieving the next and previous keys in the filter. This operation performs a *LowerBound* search on the FST to reach a leaf node. If an approximation occurs in the search (i.e., a key-byte at certain level does not exist in the trie and it has to move to the next valid label), then the function sets the *fp_flag* to false and returns the current iterator. Otherwise, the prefix of k matches that of a stored key (k') in the trie. SuRF-Base and SuRF-Hash do not have auxiliary suffix bits that can determine the order between k and k'; they have to set the *fp_flag* to true and return the iterator pointing to k'. SuRF-Real includes the real suffix bits k'_r for k' to further compare to the corresponding real suffix bits k_r for k. If $k'_r > k_r$, *fp_flag* = false and return the current iterator; If $k'_r = k_r$, *fp_flag* = true and return the current iterator; If $k'_r < k_r$, *fp_flag* = false and return the advanced iterator (iter++).

4. FST & SuRF MICROBENCHMARKS

In this section, we first evaluate SuRF and its underlying FST data structure using in-memory microbenchmarks to provide a comprehensive understanding of the filter's strengths and weaknesses. We use the Yahoo! Cloud Serving Benchmark (YCSB) [17] workloads C and E to generate point and range queries. We test two representative key types: 64-bit random integers generated by YCSB and email addresses (host reversed, e.g., "com.domain@foo") drawn from a real-world dataset (average length = 22 bytes, max length = 129 bytes). The machine on which we run the experiments has two Intel Xeon E5-2680v2 CPUs @ 2.80 GHz, 4×32 GB RAM. The experiments run on a single thread. We omit error bars because the variance is small.

4.1 FST Evaluation

We compare FST to three state-of-the-art pointer-based indexes:

- **B+tree**: This is the most common index structure used in database systems. We use the fast STX B+tree [14] with node size set to 512 bytes for best in-memory performance. We tested only with fixed-length keys (i.e., 64-bit integers).
- **ART**: The Adaptive Radix Tree (ART) is a state-of-the-art index structure designed for in-memory databases [26]. ART adaptively chooses from four different node layouts based on branching density to achieve better cache performance and space-efficiency.
- **C-ART**: We obtain a compact version of ART by constructing a plain ART instance and converting it to a static version [38].

We begin each experiment by bulk-loading a sorted key list into the index. The list contains 50M entries for the integer keys and 25M entries for the email keys. We report the average throughput of 10M point or range queries on the index. The YCSB default range queries are short: most queries scan 50–100 items, and the access

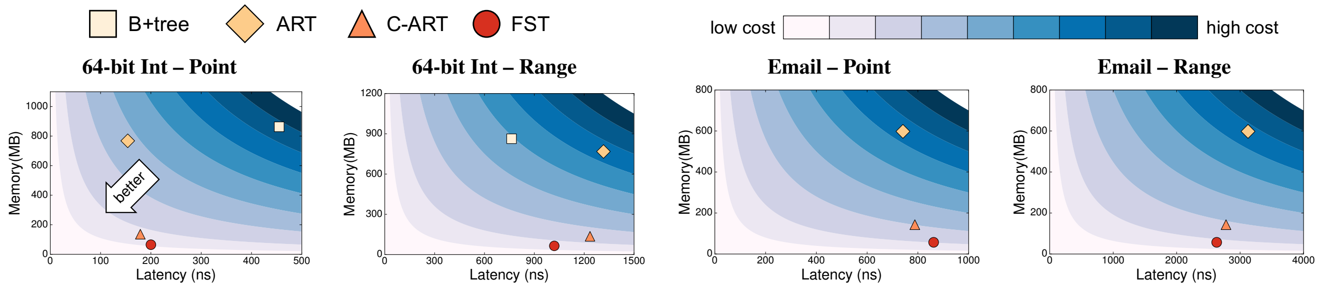


Figure 4: FST vs. Pointer-based Indexes

patterns follow a Zipf distribution. The average query latency here refers to the reciprocal of throughput because our microbenchmark executes queries serially in a single thread. For all index types, the reported memory number excludes the space taken by the value pointers.

ART, C-ART, and FST store only unique key prefixes in this experiment as described in Section 3.1. Fig. 4 shows the comparison results. Each subfigure plots the locations of the four (three for email keys) indexes in the performance-space (latency vs. memory) map. We observe that FST is among the fastest choices in all cases while consuming less space. To better understand this trade-off, we define a cost function $C = P^r S$, where P represents performance (latency), and S represents space (memory). The exponent r indicates the relative importance between P and S. $r > 1$ means that the application is performance critical, and $0 < r < 1$ suggests otherwise. We define an "indifference curve" as a set of points in the performance-space map that have the same cost. We draw the equi-cost curves in Fig. 4 using cost function $C = PS$ ($r = 1$), assuming a balanced performance-space trade-off. We observe that FST has the lowest cost (i.e., is most efficient) in all cases. In order for the second place (C-ART) to have the same cost as FST in the first subfigure, for example, r needs to be 6.7 in the cost function, indicating an extreme preference for performance.

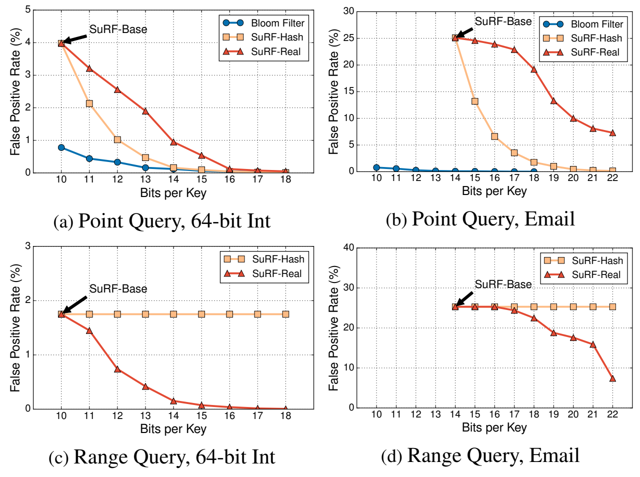
We also compared FST against other succinct trie alternatives (i.e., tx-trie [10] and the path-decomposed trie (PDT) [23]). Our results showed that FST is 6–15× faster than tx-trie, 4–8× faster than PDT, and is also smaller than both. Detailed evaluation is included in the original SIGMOD paper [39].

4.2 SuRF Evaluation

The three most important metrics with which to evaluate SuRF are false positive rate (FPR), performance, and space. The datasets are 100M 64-bit random integer keys and 25M email keys. In the experiments, we first construct the filter under test using half of the dataset selected at random. We then execute 10M point or range queries on the filter. The querying keys (K) are drawn from the *entire* dataset according to YCSB workload C so that roughly 50% of the queries return false. We tested two query access patterns: uniform and Zipf distribution. We show only the Zipf distribution results because the observations from both patterns are similar. For 64-bit random integer keys, the range query is $[K + 2^{37}, K + 2^{38}]$ where 46% of the queries return true. For email keys, the range query is [K, K(with last byte ++)] (e.g., [org.acm@sigmod, org.acm@sigmoe]) where 52% of the queries return true. We use the Bloom filter implementation from RocksDB.

4.2.1 False Positive Rate

Fig. 5 shows the false positive rate (FPR) comparison between SuRF variants and the Bloom filter by varying the size of the filters. The Bloom filter only appears in point queries. Note that SuRF-Base consumes 14 (instead of 10) bits per key for the email key



(a) Point Query, 64-bit Int (b) Point Query, Email (c) Range Query, 64-bit Int (d) Range Query, Email

Figure 5: False positive rate comparison between SuRF variants and the Bloom filter (lower is better)

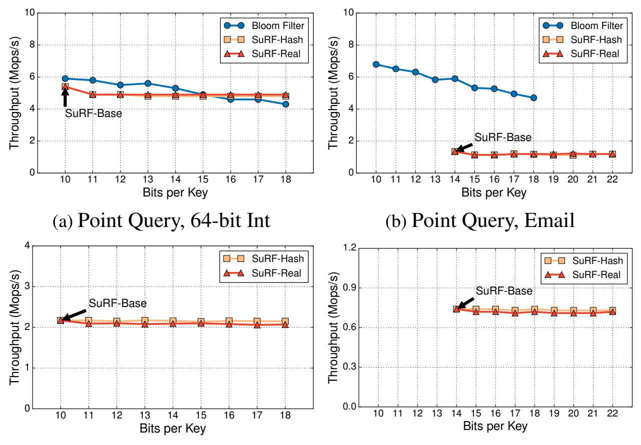
workloads. This is because email keys share longer prefixes, which increases the number of internal nodes in SuRF.

For point queries, the Bloom filter has lower FPR than the same-sized SuRF variants in most cases, although SuRF-Hash catches up quickly as the number of bits per key increases because every hash bit added cuts the FPR by half. Real suffix bits in SuRF-Real are generally less effective than hash bits for point queries. For range queries, only SuRF-Real benefits from increasing filter size because the hash suffixes in SuRF-Hash do not provide ordering information. The shape of the SuRF-Real curves in the email key workloads (i.e., the latter 4 suffix bits are more effective in recognizing false positives than the earlier 4) is because of ASCII encoding of characters.

We also observe that SuRF variants have higher FPRs for the email key workloads. This is because the email keys in the data set are very similar (i.e., the key distribution is dense). Two email keys often differ by the last byte, or one may be a prefix of the other. If one of the keys is represented in the filter and the other key is not, querying the missing key on SuRF-Base is likely to produce false positives. The high FPR for SuRF-Base is significantly lowered by adding suffix bits, as shown in the figures.

4.2.2 Performance

Fig. 6 shows the throughput comparison. The SuRF variants operate at a speed comparable to the Bloom filter for the 64-bit integer key workloads, thanks to the LOUDS-DS design and other performance optimizations such as vectorized label search and memory prefetching. For email keys, the SuRF variants are slower than the Bloom filter because of the overhead of searching/traversing the long prefixes in the trie. The Bloom filter's throughput decreases as the number of bits per key gets larger because larger Bloom fil-



(a) Point Query, 64-bit Int (b) Point Query, Email

(c) Range Query, 64-bit Int (d) Range Query, Email

Figure 6: Performance comparison between SuRF variants and the Bloom filter (higher is better)

ters require more hash probes. The throughput of the SuRF variants does not suffer from increasing the number of suffix bits because as long as the suffix length is less than 64 bits, checking with the suffix bits only involves one memory access and one integer comparison. The (slight) performance drop in the figures when adding the first suffix bit (i.e., from 10 to 11 for integer keys, and from 14 to 15 for email keys) demonstrates the overhead of the extra memory access to fetch the suffix bits. Range queries in SuRF are slower than point queries because every query needs to walk down to the bottom of the trie (no early exit).

Some high-level takeaways from the experiments: (1) SuRF can perform range filtering while the Bloom filter cannot; (2) If the target application only needs point query filtering with moderate FPR requirements, the Bloom filter is usually a better choice than SuRF; (3) For point queries, SuRF-Hash can provide similar theoretical guarantees on FPR as the Bloom filter, while the FPR for SuRF-Real depends on the key distribution.

5. EXAMPLE APPLICATION: ROCKSDB

We integrated SuRF with RocksDB as a replacement for its Bloom filter. Incoming writes go into the RocksDB's MemTable and are appended to a log file for persistence. When the MemTable is full (e.g., exceeds 4 MB), the engine sorts it and then converts it to an SSTable that becomes part of level 0. An SSTable contains sorted key-value pairs and is divided into fixed-length blocks matching the smallest disk access units. To locate blocks, RocksDB stores the "restarting point" (a string that is $\geq$ the last key in the current block and $<$ the first key in the next block) for each block as an index. When the size of a level hits a threshold, RocksDB selects an SSTable at this level and merges it into the next-level SSTables that have overlapping key ranges. This process is called compaction. Except for level 0, all SSTables at the same level have disjoint key ranges. In other words, the keys are globally sorted for each level ≥ 1. This property ensures that an entry lookup reads at most one SSTable per level for levels ≥ 1.

We modified RocksDB's point (*Get*) and range (*Seek*, *Next*) query implementations to use SuRF. For *Get(key)*, SuRF is used exactly like the Bloom filter. Specifically, RocksDB searches level by level. At each level, RocksDB locates the candidate SSTable(s) and block(s) (level 0 may have multiple candidates) via the block indexes in the table cache. For each candidate SSTable, if a filter is available, RocksDB queries the filter first and fetches the SSTable block only if the filter result is positive. If the filter result is negative, the candidate SSTable is skipped and the unnecessary I/O is saved.

For *Seek(lk, hk)*, if *hk* (high key) is not specified, we call it an *Open Seek*. Otherwise, we call it a *Closed Seek*. To implement *Seek(lk, hk)*, RocksDB first collects the candidate SSTables from all levels by searching for *lk* (low key) in the block indexes.

Absent SuRFs, RocksDB examines each candidate SSTable and fetches the block containing the smallest key that is $\geq$ *lk*. RocksDB then compares the candidate keys and finds the global smallest key $K \geq$ *lk*. For an Open Seek, the query succeeds and returns the iterators (at least one per level). For a Closed Seek, however, RocksDB performs an extra check against the *hk*: if $K \leq hk$, the query succeeds; otherwise the query returns an invalid iterator.

With SuRFs, however, instead of fetching the actual blocks, RocksDB can obtain the candidate key for each SSTable by performing a *moveToNext(lk)* query on its SuRF to avoid the one I/O per SSTable. If the query succeeds (i.e., Open Seek or $K \leq hk$), RocksDB fetches exactly one block from the selected SSTable that contains the global minimum K. If the query fails (i.e., $K > hk$), no I/O is involved. Because SuRF's moveToNext query returns only a key prefix K_p, three additional checks are required to guarantee correctness. First, if the moveToNext query sets the false positive flag, RocksDB must fetch the complete key K from the SSTable block to determine whether $K \geq lk$. If not set, RocksDB fetches the next key after K. Second, if K_p is a prefix of *hk*, the complete key K is also needed to verify $K \leq hk$. If not, the current SSTable is skipped. Third, multiple key prefixes could tie for the smallest. In this case, RocksDB must fetch their corresponding complete keys from the SSTable blocks to find the globally smallest. Despite the three potential additional checks, using SuRF in RocksDB reduces the average I/Os per *Seek(lk, hk)* query.

To illustrate how SuRFs benefit range queries, suppose a RocksDB instance has three levels (L_N, L_{N-1}, L_{N-2}) of SSTables on disk. L_N has an SSTable block containing keys 2000, 2011, 2020 with 2000 as the block index; L_{N-1} has an SSTable block containing keys 2012, 2014, 2029 with 2012 as the block index; and L_{N-2} has an SSTable block containing keys 2008, 2021, 2023 with 2008 as the block index. Consider the range query [2013, 2019]. Using only block indexes, RocksDB has to read all three blocks from disk to verify whether there are keys between 2013 and 2019. Using SuRFs eliminates the blocks in L_N and L_{N-2} because the filters for those SSTables will return false to query [2013, 2019] with high probability. The number of I/Os is likely to drop from three to one.

Next(hk) is similar to *Seek(lk, hk)*, but the iterator at each level is already initialized. RocksDB must only increment the iterator pointing to the current key, and then repeat the "find the global smallest" algorithm as in *Seek*.

6. SYSTEM EVALUATION

Time-series databases often use RocksDB or similar LSM-tree designs for the storage engine. Examples are InfluxDB [4], QuasarDB[6], LittleTable [32] and Cassandra-based systems [5, 25]. We thus create a synthetic RocksDB benchmark to model a time-series dataset generated from distributed sensors and use this for end-to-end performance measurements. We simulated 2K sensors to record events. The key for each event is a 128-bit value comprised of a 64-bit timestamp followed by a 64-bit sensor ID. The associated value in the record is 1 KB long. The occurrence of each event detected by each sensor follows a Poisson distribution with an expected frequency of one every 0.2 seconds. Each sensor operates for 10K seconds and records ~50K events. The starting timestamp for each sensor is randomly generated within the first

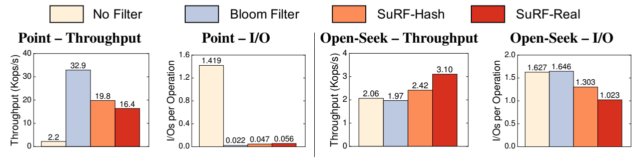


Figure 7: RocksDB point query and Open-Seek query evaluation under different filter configurations

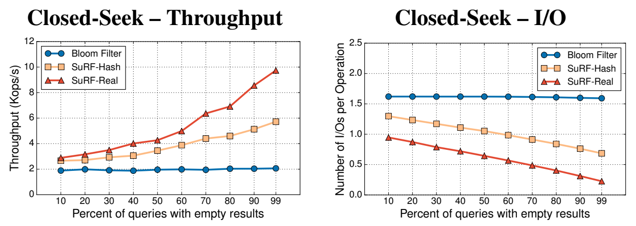


Figure 8: RocksDB Closed-Seek query evaluation under different filter configurations and range sizes

0.2 seconds. The total size of the raw records is approximately 100 GB.

Our testing framework supports the following queries:

- **Point Query**: Given a timestamp and a sensor ID, return the record if there is an event.
- **Open-Seek Query**: Given a starting timestamp, return an iterator pointing to the earliest event after that time.
- **Closed-Seek Query**: Given a time range, determine whether any events happened during that time period. If yes, return an iterator pointing to the earliest event in the range.

Our test machine has an Intel Core i7-6770HQ CPU, 32 GB RAM, and an Intel 540s 480 GB SSD. We use Snappy (RocksDB's default) for data compression. The resulting RocksDB instance has four levels (including Level 0) and uses 52 GB of disk space. We configured RocksDB according Facebook's recommendations [7, 20].

We create four instances of RocksDB with different filter options: no filter, Bloom filter, SuRF-Hash, and SuRF-Real. We configure each filter to use an equal amount of memory. Bloom filters use 14 bits per key. The equivalent-sized SuRF-Hash and SuRF-Real include a 4-bit suffix per key. We first warm the cache with 1M uniformly-distributed point queries to existing keys so that every SSTable is touched $\sim$ 1000 times and the block indexes and filters are cached. After the warm-up, both RocksDB's block cache and the OS page cache are full. We then execute 50K application queries, recording the end-to-end throughput and I/O counts. We compute the DBMS's throughput by dividing query counts by execution time, while I/O counts are read from system statistics before and after the execution. The query keys (for range queries, the starting keys) are randomly generated. The reported numbers are the average of three runs. Even though RocksDB supports prefix Bloom filters, we exclude them in our evaluation because they do not offer benefits over Bloom filters in this scenario: (1) range queries using arbitrary integers do not have pre-determined key prefixes, which makes it hard to generate such prefixes, and (2) even if key prefixes could be determined, prefix Bloom filters always return false positives for point lookups on absent keys sharing the same prefix with any present key, incurring high false positive rates.

Fig. 7 (left two figures) shows the result for point queries. Because the query keys are randomly generated, almost all queries return false. The query performance is dominated by the I/O count: they are inversely proportional. Excluding Level 0, each point query is expected to access three SSTables, one from each level (Level 1, 2, 3). Without filters, point queries incur approximately 1.5 I/Os per operation according to Fig. 7, which means that the entire Level 1 and approximately half of Level 2 are likely cached. This agrees with the typical RocksDB application setting where the last two levels are not cached in memory [19].

Using filters in point queries reduces I/O because they prevent unnecessary block retrieval. Using SuRF-Hash or SuRF-Real is slower than using the Bloom filter because the 4-bit suffix does not reduce false positives as low as the Bloom filter configuration (refer to Section 4.2.1). SuRF-Real provides similar benefit to SuRF-Hash because the key distribution is sparse.

The main benefit of using SuRF is speeding range queries. Fig. 7 (right two figures) shows that using SuRF-Real can speed up Open-Seek queries by 50%. SuRF-Real cannot improve further because an Open-Seek query requires reading at least one SSTable block as described in Section 5, and that SSTable block read is likely to occur at the last level where the data blocks are not available in cache. In fact, the I/O figure (rightmost) shows that using SuRF-Real reduces the number of I/Os per operation to 1.023, which is close to the maximum I/O reduction for Open-Seeks.

Fig. 8 shows the throughput and I/O count for Closed-Seek queries. On the x-axis, we control the percent of queries with empty results by varying the range size. The Poisson distribution of events from all sensors has an expected frequency of one per $\lambda = 10^5$ ns. For an interval with length R, the probability that the range contains no event is given by $e^{-R/\lambda}$. Therefore, for a target percentage (P) of Closed-Seek queries with empty results, we set range size to $\lambda \ln(\frac{1}{P})$. For example, for 50%, the range size is 69310 ns.

Similar to the Open-Seek query results, the Bloom filter does not help range queries and is equivalent to having no filter. Using SuRF-Real, however, speeds up the query by 5$\times$ when 99% of the queries return empty. Again, I/O count dominates performance. Without a range filter, every query must fetch candidate SSTable blocks from each level to determine whether there are keys in the range. Using the SuRF variants, however, avoids many of the unnecessary I/Os; RocksDB performs a read to the SSTable block containing that minimum key only when the minimum key returned by the filters at each level falls into the querying range. Using SuRF-Real is more effective than SuRF-Hash in this case because the real suffix bits help reduce false positives at the range boundaries.

To continue scanning after *Seek*, the DBMS calls *Next* and advances the iterator. We do not observe performance improvements for *Next* when using SuRF because the relevant SSTable blocks are already loaded in memory. Hence, SuRF mostly helps short range queries. As the range gets larger, the filtering benefit is amortized.

As a final remark, we evaluated RocksDB in a setting where the memory vs. storage budget is generous. The DBMS will benefit more from SuRF under tighter constraints and/or a larger dataset.

7. RELATED WORK

The Bloom filter [15] and its major variants [16, 21, 31] are compact data structures designed for fast approximate membership tests. They are widely used in storage systems, especially LSM trees as described in the introduction, to reduce expensive disk I/O. Similar applications can be found in distributed systems to reduce network I/O [8, 35, 37]. The downside for Bloom filters, however, is that they cannot handle range queries because their hashing does not preserve key order. In practice, people use prefix Bloom filters to help answer range-emptiness queries. For example,

Block cache size = 1 GB; OS page cache $\leq$ 3 GB. Enabled `pin_l0_filter_and_index_blocks_in_cache` and `cache_index_and_filter_blocks`.

RocksDB [2], LevelDB [3], and LittleTable [32] store pre-defined key prefixes in Bloom filters so that they can identify an empty-result query if they do not find a matching prefix in the filters. Compared to SuRFs, this approach, however, has worse filtering ability and less flexibility. It also requires additional space to support both point and range queries.

Adaptive Range Filter (ARF) [11] was introduced as part of Project Siberia in Hekaton [18] to guard cold data. An ARF is a simple encoded binary tree that covers the entire key space. ARF differs from SuRF in that it targets different applications and scalability goals. First, ARF behaves more like a cache than a general-purpose filter. It requires knowledge about prior queries for training. SuRF, on the other hand, assumes nothing about workloads. In addition, ARF's binary tree design makes it difficult to accommodate variable-length string keys because a split key that evenly divides a parent node's key space is not well defined in the variable-length string key space. In contrast, SuRF natively supports variable-length string keys with its trie design. Finally, ARF performs a linear scan over the entire level when traversing down the tree. Linear lookup complexity prevents ARF from scaling. SuRF avoids linear scans by navigating its internal tree structure with rank & select operations.

8. CONCLUSION

This paper introduces the SuRF filter structure, which supports approximate membership tests for single keys and ranges. SuRF is built upon a new succinct data structure, called the Fast Succinct Trie (FST), that requires only 10 bits per node to encode the trie. FST is engineered to have performance equivalent to state-of-the-art pointer-based indexes. SuRF is memory efficient, and its space/false positive rates can be tuned by choosing different amounts of suffix bits to include. Replacing the Bloom filters with SuRFs of the same size in RocksDB substantially reduced I/O and improved throughput for range queries with a modest cost on the worst-case point query throughput. We believe, therefore, that SuRF is a promising technique for optimizing future storage systems, and more. SuRF's source code is publicly available at `https://github.com/efficient/SuRF`.

References

[1] Facebook MyRocks. `http://myrocks.io/`.
[2] Facebook RocksDB. `http://rocksdb.org/`.
[3] Google LevelDB. `https://github.com/google/leveldb`.
[4] The influxdb storage engine and the time-structured merge tree (tsm). `https://docs.influxdata.com/influxdb/v1.0/concepts/storage_engine/`.
[5] Kairosdb. `https://kairosdb.github.io/`.
[6] Quasardb. `https://en.wikipedia.org/wiki/Quasardb`.
[7] RocksDB Tuning Guide. `https://github.com/facebook/rocksdb/wiki/RocksDB-Tuning-Guide`.
[8] Squid Web Proxy Cache. `http://www.squid-cache.org/`.
[9] Succinct data structures. `https://en.wikipedia.org/wiki/Succinct_data_structure`.
[10] tx-trie 0.18 – succinct trie implementation. `https://github.com/hillbig/tx-trie`, 2010.
[11] K. Alexiou, D. Kossmann, and P.-Å. Larson. Adaptive range filters for cold data: Avoiding trips to siberia. *Proceedings of the VLDB Endowment*, 6(14):1714–1725, 2013.
[12] D. Arroyuelo, R. Cánovas, G. Navarro, and K. Sadakane. Succinct trees in practice. In *Proceedings of ALENEX '10*, pages 84–97, 2010.
[13] D. Benoit, E. D. Demaine, J. I. Munro, R. Raman, V. Raman, and S. S. Rao. Representing trees of higher degree. *Algorithmica*, 43(4):275–292, 2005.
[14] T. Bingmann. Stx b+ tree c++ template classes. `http://idlebox.net/2007/stx-btree/`, 2008.
[15] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
[16] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese. An improved construction for counting bloom filters. In *European Symposium on Algorithms*, pages 684–695. Springer, 2006.
[17] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of SOCC'10*, pages 143–154. ACM, 2010.
[18] C. Diaconu, C. Freedman, E. Ismert, P.-A. Larson, P. Mittal, R. Stonecipher, N. Verma, and M. Zwilling. Hekaton: Sql server's memory-optimized oltp engine. In *Proceedings of SIGMOD'13*, pages 1243–1254. ACM, 2013.
[19] S. Dong. personal communication, 2017. 2017-08-28.
[20] S. Dong, M. Callaghan, L. Galanis, D. Borthakur, T. Savor, and M. Strum. Optimizing space amplification in rocksdb. In *CIDR*, volume 3, page 3, 2017.
[21] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking (TON)*, 8(3):281–293, 2000.
[22] R. González, S. Grabowski, V. Mäkinen, and G. Navarro. Practical implementation of rank and select queries. In *Proceedings of WEA'05*, pages 27–38, 2005.
[23] R. Grossi and G. Ottaviano. Fast compressed tries through path decompositions. *Journal of Experimental Algorithmics (JEA)*, 19:3–4, 2015.
[24] G. Jacobson. Space-efficient static trees and graphs. In *Foundations of Computer Science*, pages 549–554. IEEE, 1989.
[25] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.
[26] V. Leis, A. Kemper, and T. Neumann. The adaptive radix tree: Artful indexing for main-memory databases. In *ICDE'13*, pages 38–49. IEEE, 2013.
[27] M. Martínez-Prieto, N. Brisaboa, R. Cánovas, F. Claude, and G. Navarro. Practical compressed string dictionaries. *Information Systems*, 56:73–108, 2016.
[28] J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing*, 31(3):762–776, 2001.
[29] G. Navarro and E. Providel. Fast, small, simple rank/select on bitmaps. In *Proceedings of SEA '12*, pages 295–306, 2012.
[30] P. O'Neil, E. Cheng, D. Gawlick, and E. O'Neil. The log-structured merge-tree (lsm-tree). *Acta Informatica*, 33(4):351–385, 1996.
[31] F. Putze, P. Sanders, and J. Singler. Cache-, hash-and space-efficient bloom filters. In *Proceedings of WEA'07*, pages 108–121. Springer, 2007.
[32] S. Rhea, E. Wang, E. Wong, E. Atkins, and N. Storer. Littletable: a time-series database and its uses. In *Proceedings of SIGMOD'17*, pages 125–138. ACM, 2017.
[33] K. Sadakane and G. Navarro. Fully-functional succinct trees. In *SODA'10*, 2010.
[34] R. Sears and R. Ramakrishnan. blsm: a general purpose log structured merge tree. In *Proceedings of SIGMOD'12*, pages 217–228. ACM, 2012.
[35] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood. Fast hash table lookup using extended bloom filter: an aid to network processing. *ACM SIGCOMM Computer Communication Review*, 35(4):181–192, 2005.
[36] S. Vigna. Broadword implementation of rank/select queries. In *Proceedings of WEA'08*, pages 154–168, 2008.
[37] M. Yu, A. Fabrikant, and J. Rexford. Buffalo: Bloom filter forwarding architecture for large organizations. In *Proceedings of CoNEXT'09*, pages 313–324. ACM, 2009.
[38] H. Zhang, D. G. Andersen, A. Pavlo, M. Kaminsky, L. Ma, and R. Shen. Reducing the storage overhead of main-memory oltp databases with hybrid indexes. In *Proceedings of SIGMOD'16*, pages 1567–1581. ACM, 2016.
[39] H. Zhang, H. Lim, V. Leis, D. G. Andersen, M. Kaminsky, K. Keeton, and A. Pavlo. Surf: practical range query filtering with fast succinct tries. In *Proceedings of SIGMOD'18*, pages 323–336. ACM, 2018.
[40] D. Zhou, D. G. Andersen, and M. Kaminsky. Space-efficient, high-performance rank and select structures on uncompressed bit sequences. In *Proceedings of SEA '13*, pages 151–163. Springer, 2013.